Advance NBA Scouting Report

Ron Kinel, Kenneth Liu, Zihao Niu, Derek Yan
UCSB
December 2019

Instructor: Oh Sang-Yung PhD

Table of Contents

Table of Figures

Abstract

In this project we use Python to summarize and analyze the data of an opposing team for the next NBA game on a selected team's schedule. This report is a real time report which factors in all games the opposing team has played up to that game. For any selected team, the final result of the project will allow the coach for that selected team to just input the game number and output a fully detailed scouting report of the opposing team. First, we used NBA API in Python to pull all team schedules and some additional statistics. We then loaded all shots data, as used in assignment 2 in PSTAT 134. We used the 2018-19 NBA season since it has been completed but this could be used for the ongoing season as well, since our report only uses games that have been played before the desired game. Next, we created functions that pull the opponent players shots and rebounds from all previous games. We created functions that pull the team name and record and determine whether this game is a 2nd night of a back-to-back for the opposing team. We created tables and graphs to analyze the player's and their team's tendencies in the whole game and in close games as well. Finally, we evaluate our report as a whole and the insight it provides and how it could be improved in the future.

Background

The NBA regular season schedule is a long 82 game season, in which coaches must maximize the number of wins while resting players and playing as efficiently as possible, especially if they have playoff aspirations. Scouting reports are very important to the performance of basketball teams because they analyze the strengths and weaknesses of the opposing team and advise the coach on how to deal with them, helping him or her achieve an efficient gameplan. If a team wants to find a player who has high potential in a game against them, the scouting report is one of the most important tools they have. A good scouting report will analyze different players' performances throughout the game in different circumstances. It will also analyze different overall team tendencies. Numbers such as top rebounders, best shooters, etc. at different times of the game are very insightful and are desired for our program.

Scouting reports have been around in the NBA for a long time, however, traditionally they were done by watching films, something that took a lot of time and money. Staff on the team would watch the film and keep certain footage that may be useful for the rest of the team and coaches to watch. Today, film sessions are still popular in the NBA but both team tendencies and scouting reports have become more and more statistically driven. There are limitations to what statistics can pull, and film sometimes is required for things such as defensive schemes or offensive plays. However, statistics are playing a larger role in the field, and we will try to incorporate it into our report.

Objective

The objective of this project is to help coaches, by creating a program that could numeralize the actual performance and behavior of top players, to find the players who have the highest efficiency, most eager to win and can bear the most pressure while they are influenced by several environmental factors in a game such as the period, whether it is a close-game, or a 2nd night of back-to-back. We will use the data to calculate the efficiency of each player and provide additional analysis to top players. Since some players speciality is not scoring, we will also provide some analysis for rebounding. Lastly, we want to create a close game report, analyzing the team tendencies in the close games they've had so far in the season. The end goal, is a cohesive report, providing a coach many of the details he needs to come up with an efficient and productive game plan, especially defensively.

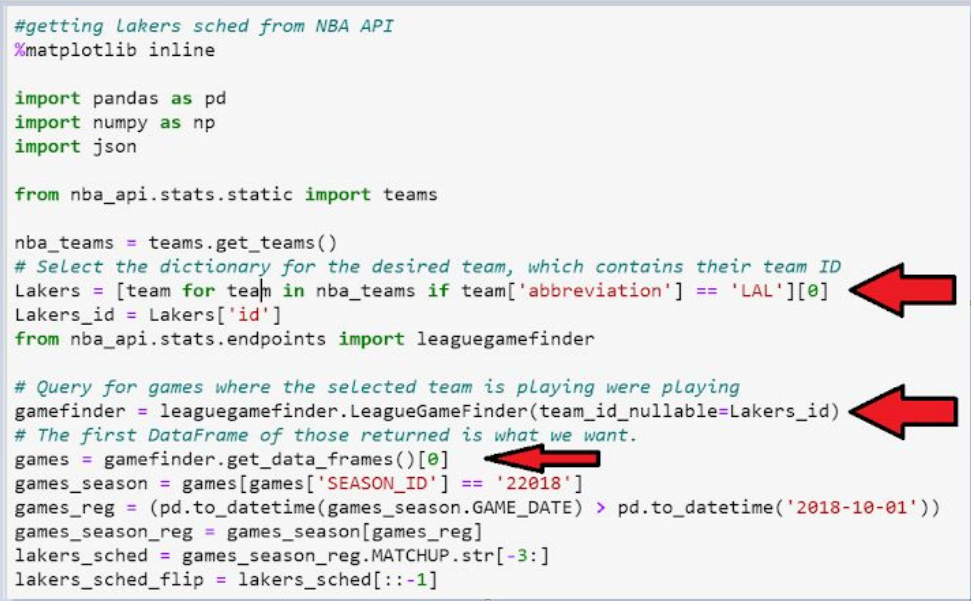
Data

This project was extremely data driven and thus a lot of time was spent processing data. Since this was a large step in our project, we will discuss the process in a detailed fashion during this section of the report.

We pulled NBA data in two different ways. First, using the NBA API in python we pulled data from the NBA website containing team schedules, including game results for played games. Additionally, we pulled player statistics per game for rebounding information. Next, we used the ucsb box link from PSTAT 134's assignment 2 to get all shots data in the NBA, a dataset of every shot taken in the NBA.

Familiarizing ourselves and processing all the data was a timely task. We first wrote a code chunk, where the desired team who one is coaching for is selected, and the team game schedule for the 2018-19 season is pulled out; this can be changed to any season fairly easily, including the current NBA season. It is additionally very easy to choose the desired team, with just the team name needed; for the entirety of our report the desired team was the Los Angeles Lakers (LAL), but again this could easily be changed to any other team. We did this by extracting the team's ID using the 3 letter team acronym so that we can find all the games that the team will play this season. As can be seen in *figure 1* below, this was not an easy task as the NBA API data frame is in a unique structure and some research was necessary until we were able to achieve this. However, understanding this process was crucial as it will later be used for opponent schedules as well.

Figure 1: Extracting NBA Schedules



```
#getting lakers sched from NBA API
%matplotlib inline

import pandas as pd
import numpy as np
import json

from nba_api.stats.static import teams

nba_teams = teams.get_teams()
# Select the dictionary for the desired team, which contains their team ID
Lakers = [team for team in nba_teams if team['abbreviation'] == 'LAL'][0]
Lakers_id = Lakers['id']
from nba_api.stats.endpoints import leaguegamefinder

# Query for games where the selected team is playing were playing
gamefinder = leaguegamefinder.LeagueGameFinder(team_id_nullable=Lakers_id)
# The first DataFrame of those returned is what we want.
games = gamefinder.get_data_frames()[0]
games_season = games[games['SEASON_ID'] == '22018']
games_reg = (pd.to_datetime(games_season.GAME_DATE) > pd.to_datetime('2018-10-01'))
games_season_reg = games_season[games_reg]
lakers_sched = games_season_reg.MATCHUP.str[-3:]
lakers_sched_flip = lakers_sched[::-1]
```

The arrows in *figure 1.* above point to some of the more challenging points in the chunk, where research was needed to be done since the data frame was designed in a not-so intuitive way. The main challenges were understanding initially how to extract the desired NBA data, and then pulling the selected team into its own data frame. The first arrow is pointing to the location where the desired team name should be inputed. The fifth line from the end is where the desired season ID is inputed. A date filter can be seen one line below, this was done to eliminate preseason games which are usually not very meaningful.

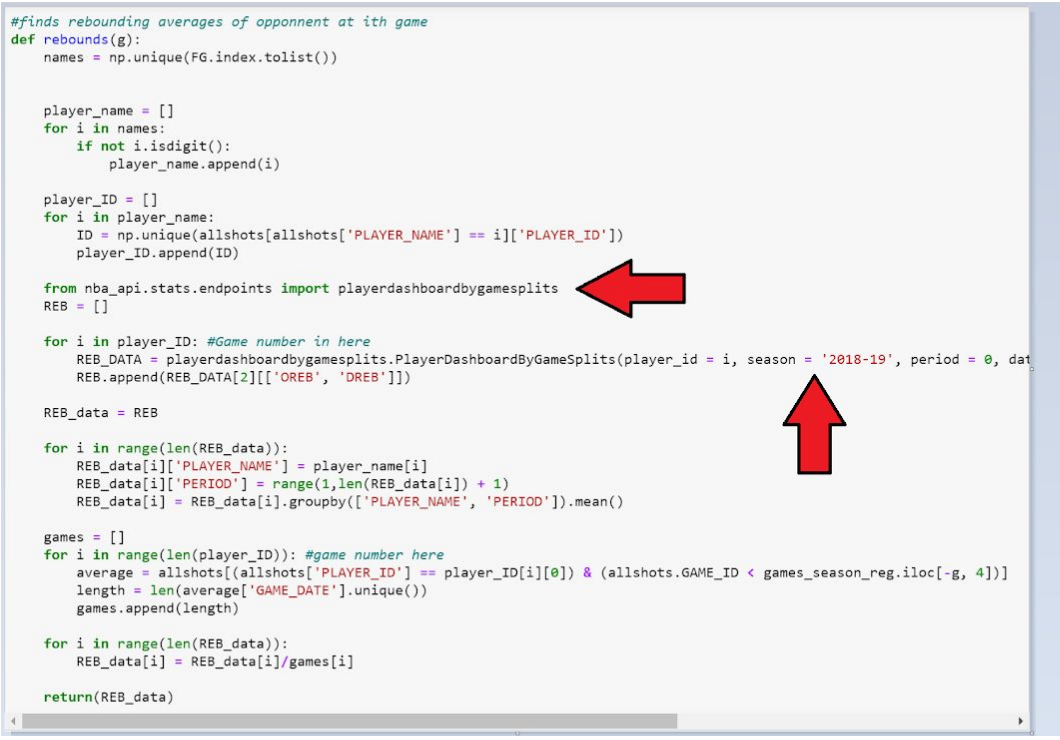
Once a desired team has been selected, the desired game number needs to be inputed. This is the game the coach wants a scouting report for. First, we had to figure out how to make the game number pull the correct team name from the desired team's schedule we previously extracted. As can be seen in *figure 1*, we did this by flipping the schedule, and having the first game of the season at the top of the dataframe. This allowed for easy indexing, pulling the correct team name using i-1, since dataframe indexing starts at 0.

Now that we have an easily accessible team schedule, we need to pull all relevant shot data for the opposing team for this game. The challenge here was to make sure we are only using data from games occurring before the desired game. A useful variable we found here is the game ID, this is because game ID's increase with every game and thus a simple less-than line was used on all shots to only select relevant ones. Another challenge was that only team ID was used in the all shots database however on the lakers schedule only opposing team name was included, thus

we needed to use a 3rd dataset to match team names with team ID, so we can pull all the shots taken by opposing team. This was done in a long line of code in the "relevant" function, see appendix B.

So far we talked about extracting all NBA shot data, filtering our only relevant shots and NBA schedules for all teams, with our desired team schedule being pulled out. Before we discuss how we process this data, we needed rebounding information for the opposing team from all previously played games. *Figure 2.* Below is our function definition for pulling out relevant rebounding information for the g^{th} game.

Figure 2: Rebounding Function



```
#finds rebounding averages of opponnent at ith game
def rebounds(g):
    names = np.unique(FG.index.tolist())


    player_name = []
    for i in names:
        if not i.isdigit():
            player_name.append(i)

    player_ID = []
    for i in player_name:
        ID = np.unique(allshots[allshots['PLAYER_NAME'] == i]['PLAYER_ID'])
        player_ID.append(ID)

    from nba_api.stats.endpoints import playerdashboardbygamesplits
    REB = []

    for i in player_ID: #Game number in here
        REB_DATA = playerdashboardbygamesplits.PlayerDashboardByGameSplits(player_id = i, season = '2018-19', period = 0, dat
        REB.append(REB_DATA[2][['OREB', 'DREB']])

    REB_data = REB

    for i in range(len(REB_data)):
        REB_data[i]['PLAYER_NAME'] = player_name[i]
        REB_data[i]['PERIOD'] = range(1,len(REB_data[i]) + 1)
        REB_data[i] = REB_data[i].groupby(['PLAYER_NAME', 'PERIOD']).mean()

    games = []
    for i in range(len(player_ID)): #game number here
        average = allshots[(allshots['PLAYER_ID'] == player_ID[i][0]) & (allshots.GAME_ID < games_season_reg.iloc[-g, 4])]
        length = len(average['GAME_DATE'].unique())
        games.append(length)

    for i in range(len(REB_data)):
        REB_data[i] = REB_data[i]/games[i]

    return(REB_data)
```

Again, the arrows in *Figure 2* above point to the challenging portions of the rebounds function. As shown by the first arrow, once we obtained the desired players' ID, we had to find out what to import from NBA API to obtain those players rebounding information. Next using the line pointed to by the second arrow, we use the NBA package we imported to obtain players rebounding totals, and the entire line can be seen in Appendix B. Something challenging about this line was again only selecting games that occurred before the desired game. This was done in the PlayerDashboardByGameSplits command which had a function variable, date_to_nullable - which took us time to find out - only selects rebounds before the given date. Lastly, we used all shots data again to count the number of games it took to reach those rebounding totals so we can

obtain the average. A key to the average above is that some players don't play in all games due to coaches' decisions or injury, and so we had to count games played by each player individually. Finally, we have obtained relevant shot information for a desired game number, NBA schedules for all 30 teams, with our desired team schedule already pulled out, and relevant rebounding averages for the desired game. We can finally move to using this data to analyze opposing team tendencies and information by building different figures and charts we found useful.

Analysis/Interpretation

The first step of our analysis was extracting the opposing team record. This was a simple step, but the team record is a basic statistic that can provide some insight on the overall level of the team being played. Next, we wanted to let our coach know about the state of the opposing team. In the NBA, games are sometimes played one day after another. This is very tiring for a team, and they will likely be more fatigued on the 2nd night of a back-to-back. Thus, we created a function in our report that informs the coach if the opposing team is playing the second game of a back-to-back against them; letting him know how tired he should expect opponents to be. This is useful because the coach can exploit the opposing team by increasing the tempo of the game.

Next, we wanted to analyze a team's offensive abilities. We decided that shooting is the key action in basketball scoring, and is the main offensive component we will be analyzing. Therefore, we first wanted to find the field goal percentage for each player on the opposing team. Field goal percentage (FG%) is the percentage of shots a player took that he made. This is a key figure in analyzing which players on the other team a coach wants to shoot the ball. The player with the lowest field goal percentage is many times that guy, which is the most likely to miss his shots.

To find the field goal percentage for each player, we used the all shot dataset described in the data section, for all games occuring before the desired game. We used a variable called Shot Made Flag, where a 1 means a shot was taken and made, and 0 means a shot was taken but missed. We simply summed all shots made flags for each player and divided by the total number of shots he took to compute the players' field goal percentage for this player. This number felt a little too broad since player's and team's actions change throughout the game, and thus we decided to give our analysis per quarter. Thus, we calculated each players' field goal percentage per quarter. This allows our team's coach to analyze a player's shot efficiency throughout the game, possibly getting tired or taking worst shots at different points of the game.

We felt that field goal percentage can be kind of a meaningless figure without getting the players' involvement in the game. If a player shoots a small number of shots, a good field goal percentage is less impactful. Thus, we created an involvement variable, this is the average

percentage of total team shots per quarter that a player takes. Now the coach knows which players are taking more shots and scoring them, meaning they should be defended closely. He can see when a player takes a lot of shots, but has low FG%, meaning this is the player you want your team to force the ball too. The coach should now have an idea of what players he wants to closely guard each quarter and what players he wants to get the ball to since they shoot and miss a lot.

Some players can contribute to the outcome of the game without shooting or scoring. Therefore we wanted to have another statistic when analyzing players' performance. We felt that rebounding information, especially offensive rebounds, will allow a coach to see the additional impact of players, especially if they don't shoot as much. So, in addition to FG% and the proportion of team shots taken, we extracted average defensive and offensive rebounding per quarter per player per game. This was done using the rebounding function described in the data portion. A player with a lot of offensive rebounds is a raised flag and the coach will likely instruct his team to pay close attention to the box out of this player. Defensive rebounds are less meaningful in the context of our analysis, but if all opposing players on the court have low defensive rebounding, the coach might want his team to crush the offensive boards more, trying to secure a rebound following a missed shot on offense.

We finally combined all these statistics into a master data frame. Since a portion of the class deals with visual representation, we felt that printing a basic pandas/python table would not be appropriate. We created a function that takes in our player statistics data frame and returns a nice looking table (Appendix B or Attached .ipynb file). In *figure 3* below, the top portion of the table is attached, describing the per quarter statistics for the first 3 players in the table.

Figure 3: Snapshot of Player Statistic Table

PLAYER_NAME	PERIOD	FG %	% of team-shots taken in q	OREB/period	DREB/period
Bogdan Bogdanovic	1	0.4493	0.0842	0.1364	0.8636
	2	0.5536	0.0724	0.1818	0.8636
	3	0.3151	0.0925	0.0909	0.3636
	4	0.4324	0.1025	0.1364	0.7273
Buddy Hield	1	0.5	0.2002	0.4118	1.0294
	2	0.439	0.1591	0.4706	0.9412
	3	0.48	0.1901	0.2059	1.3235
	4	0.4495	0.151	0.1176	0.8235
De'Aaron Fox	1	0.486	0.1306	0.2059	0.7353
	2	0.4483	0.1501	0.1765	0.8529
	3	0.5214	0.1774	0.1471	0.8235
	4	0.4255	0.1302	0.0882	0.7059

In *figure 3* above, only a snapshot of the table is included. In our actual scouting report (appendix A) all opposing players are included. It is important to mention that a period greater

than 4 is possible due to the way overtime games are coded. The players above are all guards and thus have low rebounding numbers. However, their shooting and offensive performances throughout the game can easily be analyzed. Buddy Hield is clearly the most impactful player out of the 3, especially during the 1st qtr; Shooting the most shots and making the highest field goal percentage. This analysis can be made on a quarter by quarter basis, comparing different players at different times of the game, and is much more insightful when all players are included.

Since the full table can be a bit overwhelming in terms of the amount of information it gives, we decided to pull out the top two shooting players for each quarter of regulation (only 1st 4 quarters). We also provide additional analysis for those players which we will explain soon, but simply pulling them out tells the coach who he wants the defense to pay more attention to. To decide on the top two players per quarter we decided to combine the FG% with the percentage of team shots taken. By multiplying these two statistics, we can obtain an efficiency measure that is very unique. The thought process behind this efficiency measure is charted below in *Figure 4*.

Figure 4: Efficiency Variable Logic

FG% * Proportion of Shots = Efficiency
If both numbers are high → efficiency is highest
If only one of the the two numbers are high → efficiency is medium
If both numbers are low → efficiency is lowest

We considered optimizing this efficiency by adding weights on FG% or proportion of shots but ultimately decided against it as we didn't have a response variable to truly identify players efficiency. Additionally, more statistics such as: plus/minus, turnovers, etc. could be included to improve the efficiency measure. Overall, looking at top player charts and other player information, we felt that the efficiency variable still did a very good job at including the top player per quarter.

After selecting the top 2 players per quarter using the efficiency variable described above, we wanted to provide more insightful figures regarding those players' behavior. To do this we used a similar procedure to that used in assignment 2 (Appendix B or Attached .ipynb file). *Figure 5* below shows the hot zones for the top 2 players, where they made the most shots from. For report layout reasons, only the first two quarters are included; but for the scouting report, all 4 quarters are outputted (Appendix A). With overtime periods being excluded as a different analysis approach was made for close games which went to overtime.

Figure 5: Quarter 1 and 2 Top Players' Shooting Tendencies

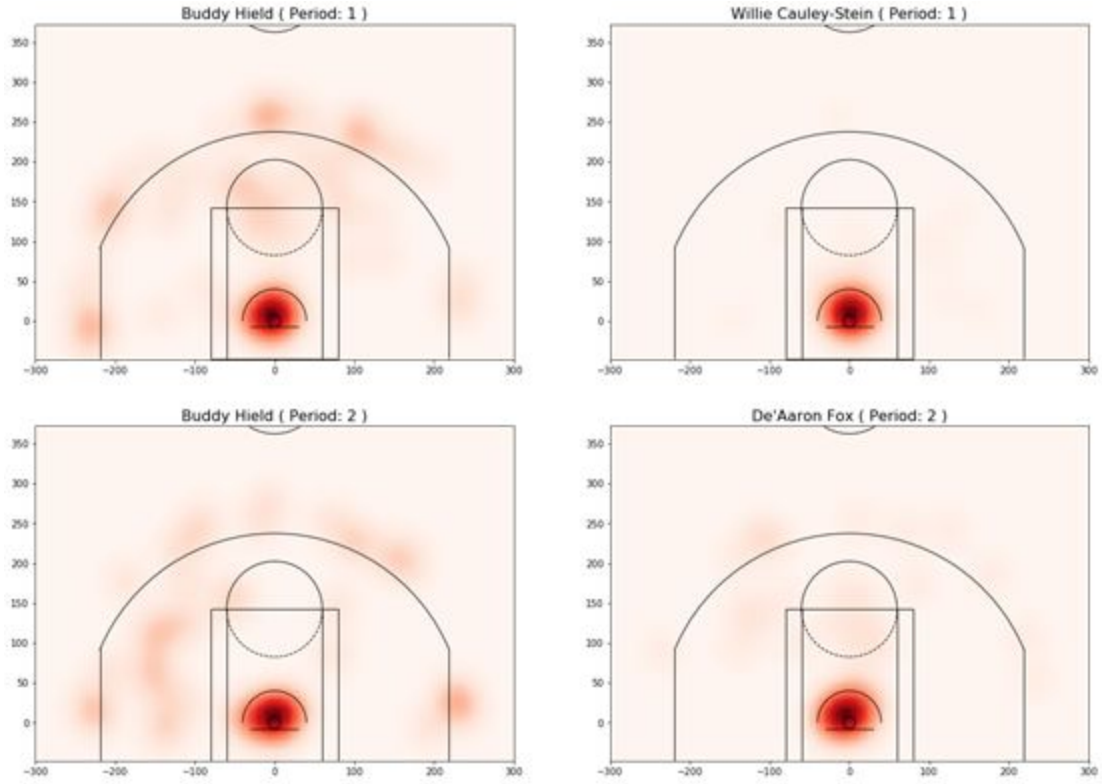


Figure 5 above provides much insight into the tendencies of top players we want to better defend. During the 1st quarter, buddy hield takes 3's from all around the arch, but many of his none-three shots are from in front of the basket or right underneath. However, in the second quarter, the difference is clear, and more of his shots inside the arch are made from the right side of the court (facing the basket). This is very reasonable as Buddy is right-handed, and also likely changes his behavior from quarter to quarter as we hypothesized. Cauley-Stein, who is a center, unsurprisingly takes all of his shots from underneath the basket. It is interesting to see that buddy hield is the top performer in both quarters but De'Aaron Fox is better than Cauley-Stein in the second quarter, meaning the coach defensive priorities should change for this quarter. Fox seems to have a slight affinity for the right side, which again makes sense since he is right-handed. Also, De'Aaron Fox is not a great three-point shooter which again can be seen by his made shot chart above. Thus, if the team is playing man-to-man defense, the coach can tell the man defending Fox to back off and give him the 3pt shot, especially in the corners. This allows the defender to play more help defense or be in a better position to grab a defensive rebound.

We faced multiple challenges when creating this figure. First, we had to find the best sigma to input into the bin shots function. Since at the beginning of the season there is a lot less data than

later in the season. A higher sigma led to better visualization of shots later in the season when more games had been played, while a lower sigma led to better visualization earlier in the season when not as many shots had been taken. We again attempted to tackle this issue with an optimization solution, but since visual quality is hard to quantify and 82 games is a lot of variation, we found that a fixed sigma actually performed better for the majority of games, while a variable sigma did not do as good of a job for a lot of games. Thus, we selected sigma to equal 5, which seemed to have given good visual results for most games, especially passed the first 5 games of the season, when top players started having a higher number of shots to analyze.

The last analysis we wanted to provide is insight for close games, i.e., how does the opposing team act on offense during the end of close games. We first needed to define what makes a close game; using a statistic called plus/minus lets us know how many points the team lost or won by. We felt that games ending by a margin of 5 points were clearly close. Additionally, if a game went into overtime, regardless of how many points the margin was, that game was a close one. Using these constraints, we selected shots taken in close games after the 5 minute mark of the 4th quarter, including over time ("Crunch time"). Analyzing players' behavior in the "crunch time" of close games will help a coach decide on defensive schemes if the end of the game is close.

We created a similar table to *figure 3* for close games. However we did not group by period since some close games might go to double over time while others are close but end in regulation. We wanted to analyze overall player performance under close game circumstances. Thus, the final table, as seen in *figure 6*, is a little simpler, so a coach can analyze the entire team's late game performance easily.

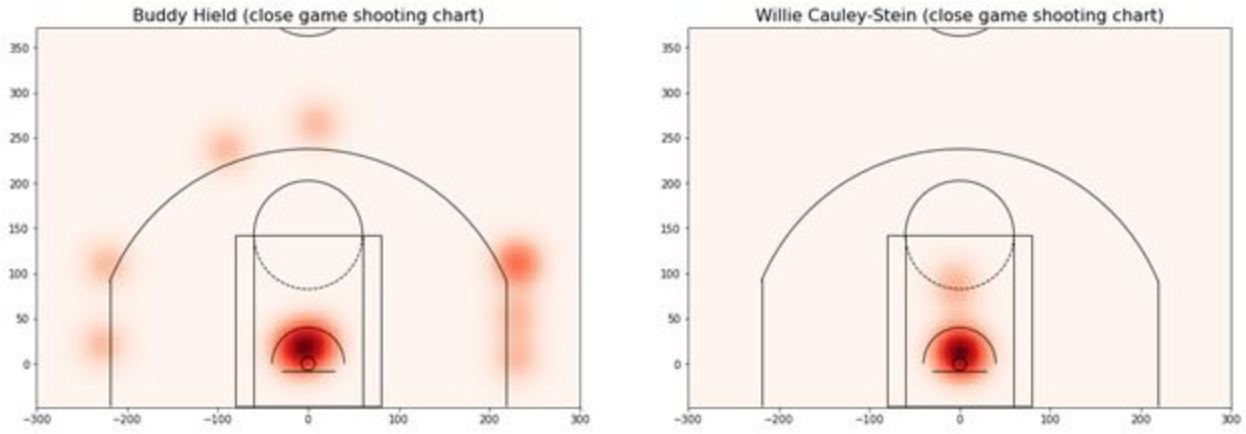
Figure 6: Close Game Player Performance

PLAYER_NAME	FG %	% of team-shots taken
Bogdan Bogdanovic	0.4286	0.0722
Buddy Hield	0.4333	0.3093
De'Aaron Fox	0.3478	0.2371
Iman Shumpert	0.5	0.0206
Justin Jackson	0.1667	0.0619
Kosta Koufos	0.0	0.0206
Marvin Bagley III	0.5	0.0619
Nemanja Bjelica	0.75	0.0412
Willie Cauley-Stein	0.5625	0.1649
Yogi Ferrell	0.0	0.0103

We find the usefulness of the table in *figure 6* to be pretty impressive. It can easily be seen that Buddy Hield is the first option during the crunch-time, taking almost a third of teams shots and knocking them with a relatively high rate. Fox is the 2nd option, taking a little less than a quarter of the shots, but his field goal percentage is significantly lower. While Willie Cauley-Stein is the 3rd option, knocking down a very high amount of his shots. In a perfect situation, there are other player's we rather take crunch-time shots for close games, but since usually you have to choose the lesser of two evils, it seems like a good game plan to get Fox to take late game shots, since it seems his team is comfortable with him taking them although he shoots at a poor percentage. There are other players with low FG% you want taking late game shots, such as Ferrell or Kosta Koufos; but looking at the numbers, those players rarely take the shot. Additionally, you don't want players with high FG% for close game crunch time shooting as well. For example, you don't want Bjelicia taking a late-game shot as he knocks down 75% of those, even though he doesn't take as many shots in late game situations.

Similarly to the full game report, we extracted the shooting charts for the two most efficient players in the crunch time of close games. This allows the coach to try and defend those players correctly in the last 5 minutes of the game and maybe avoid the game from getting any closer or going into overtime. The charts can be seen in *figure 7* below and follow a similar procedure to *figure 5*.

Figure 7: Close Game Shooting Charts



In addition to late game defensive schemes, *Figure 7* can be used to effectively defend against specific big plays. It is common in the last 2 minutes of basketball for a team to have a huge possession, where if they score or not can determine the outcome of the game; especially if it is the last possession or two. Thus, this figure can be used to tell players how to defend during a time out. Looking at *figure 7* above, you want to force Buddy Hield to take a mid-range shot, which he rarely makes in the end of close games. You don't want to let Cauley-Stein catch the

ball in the paint since he makes all of his shots from there; surprisingly not only under the basket during the end of game. If the game is on the line, you want to tell his defender to stick with him all the way up to the free throw line since he will make shots from out there.

Many times players are injured or resting and a team roster may be different than usual. Additionally, a lot of trades may happen around mid-season which would impact the effectiveness of the report right after those trades. Therefore, we created a figure with all the shots that a team made. This will help our coach understand the overall offensive tendencies of the opposing team; especially their team philosophy regarding shooting 3's as most shots in the NBA are taken from the paint or from 3pt range. For example, regardless of the player, due to their offensive schemes the Houston Rockets take a lot more 3pt shots than a team like the Kings, which is more focused on driving the ball (Appendix A).

With the inclusion of the close game shooting charts, a coach can finally get an understanding of opponents' offensive performance throughout the entire game. He should know how the defense should operate throughout the game as a whole, and how to defend the most threatening players. He has an idea of who is a threatening rebounder and which players are a threat late in close games. Our report is a quick and automated way to build very strong defensive schemes.

Conclusion

The advanced NBA scouting report was done in python and is an automated report which only uses the game number as input to provide multiple insightful figures and tables for the coach of a selected team; helping him to prepare a game plan against the opponent for that game. Using shot and game data only from applicable games, pulling the best player's from that opposing team may not seem hard to do on a case-by-case basis but it is very time consuming that way. Automating this process is a difficult task, but it saves a lot of time and makes the tool extremely easy to use. Our statistical analysis is not over-complicated, yet still combines features and provides extremely insightful figures for top player analysis and an entire team performance. We encourage you to run our report for at least different game numbers, and see the behavior differences of different teams.

Overall, the report covers some common statistics of players such as field goal percentage and rebounding information. However, these numbers are unique as they are provided in a unique per quarter grouping. Additionally insightful yet simple variables, such as proportion of shots, were created using the data. The scouting report project that we created will be able to give a quick and relevant representation and of how good the team is offensively and analyze individual players' performances throughout the game as well. One of the main challenges of this project was pulling and filtering the data from the NBA API website and processing the data so that it is

relevant to the type of project we created. We were able to come up with insightful interpretations of our figures and extract some simple but unique characteristics about the game, such as whether the game is back-to-back and the opposing team name and record.

Steps for Improvement

Rosters in the NBA sometimes change throughout the season. Players get traded or injured, changing the team dynamic. More overall team statistics could have been included to analyze an overall team behavior with no respect to the players who are playing. This may help provide an overall idea of the opposing coach's game plan, and what the opposing team is good at as a unit. Team statistics can also be more useful for defensive analysis, where the team aspect of NBA basketball is more prevalent.

In this report we chose to analyze scoring and rebounding. However, these are not the only two statistics that determine a player's quality and contribution. Assists, turnovers, steals, and blocks are all also crucial numbers for the winning of the team. We primarily focused on offense, but scoring is not the only way to view players' contributions. Steve Nash, who is a well regarded passer, didn't score much but created so many chances for his teammates to take great shots and score, which gained him the most valuable player award. So, adding more data like assist and block, in our project would give more insight to our result.

This project mainly factored in the visual representation features of this class. More validation could be done to show the useability of our scouting report. Since the result we got from the project only shows the efficiency of a player, but has nothing to say about the relation between efficiency and winning, so by validating our report we would need to show that when high efficiency players were well defended that led to high winning probability.

Although the figures are created automatically, they still need to be manually pulled from python to be visually pleasing. To improve our product, we wanted to have an automated PDF report that pulls figures to the correct location. This proved to be pretty challenging and time consuming and so we decided it will be a step for improvement.

Lastly, since this is an offensive focused report, it is important to mention that we were not able to factor fouls into our analysis. Fouls are a big part of the NBA offensive tendencies and some players may be better or worse at drawing fouls and getting points at the free throw line. There is less a coach can do to stop a foul from occurring and so it is not as crucial for a scouting report.

References

"Export a Pandas Dataframe as a Table Image." *Stack Overflow*, 31 Oct. 2014, stackoverflow.com/questions/26678467/export-a-pandas-dataframe-as-a-table-image.

"How to Save a Pandas DataFrame Table as a Png." *Stack Overflow*, 25 Feb. 2016, stackoverflow.com/questions/35634238/how-to-save-a-pandas-dataframe-table-as-a-png.

"NBA Stats." *NBA Stats*, stats.nba.com/.

Sources (for data)

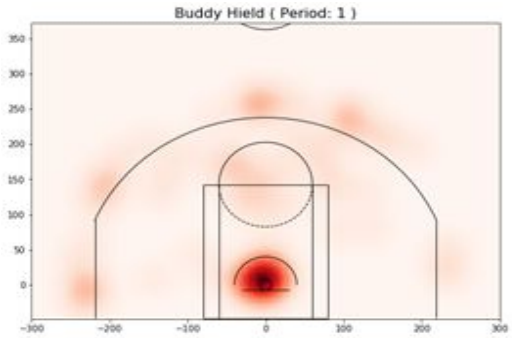
1. Shot data from URL: https://ucsb.box.com/shared/static/940qiuxyp798gv4nx8iwvazu9qqjs37e.zip
2. 'playerdashboardbygamesplits' API from nba_api.stats.endpoints for Rebound data.

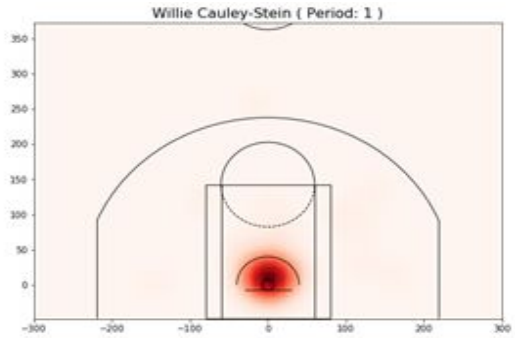
Appendices

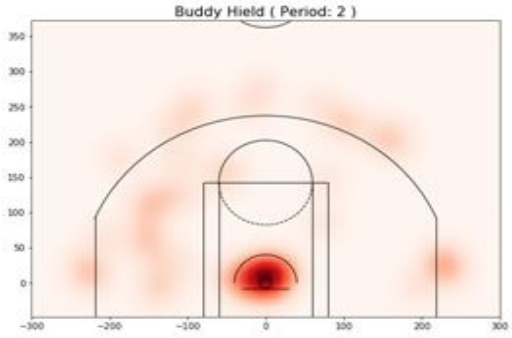
Appendix A - Sample Scouting Report: Lakers' Game Number 35

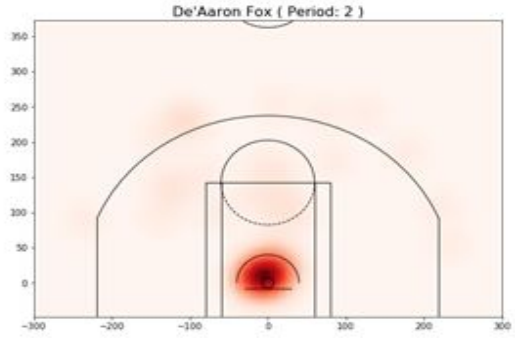
Opponent: Sacramento Kings (19-18) 2nd night of back-to-back

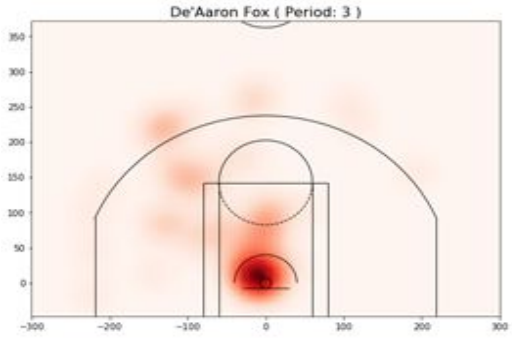
Top 2 players shooting charts by quarter:

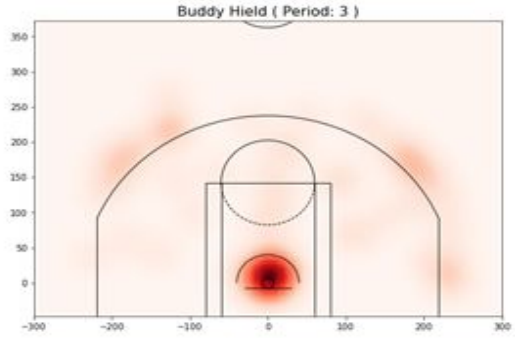


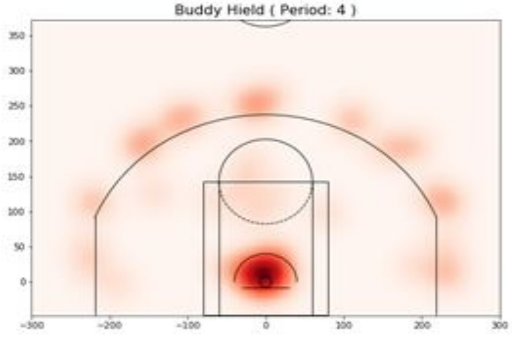


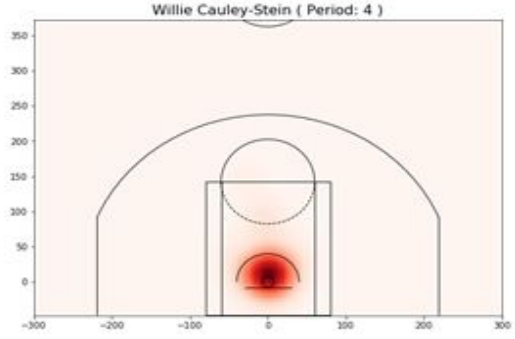












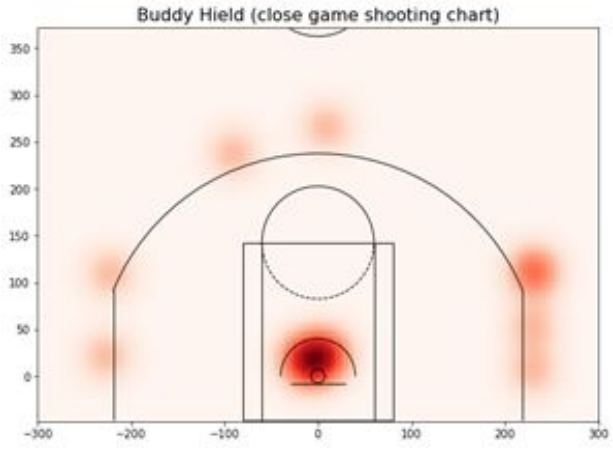
Player Performance per Qtr.:

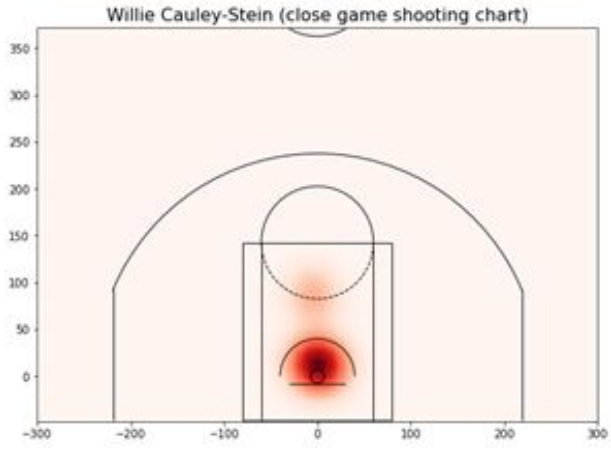
PLAYER_NAME	PERIOD	FG %	% of team-shots taken in q	OREB/period	DREB/period
Bogdan Bogdanovic	1	0.4493	0.0842	0.1364	0.8636
	2	0.5536	0.0724	0.1818	0.8636
	3	0.3151	0.0925	0.0909	0.3636
	4	0.4324	0.1025	0.1364	0.7273
Buddy Hield	1	0.5	0.2002	0.4118	1.0294
	2	0.439	0.1591	0.4706	0.9412
	3	0.48	0.1901	0.2059	1.3235
	4	0.4495	0.151	0.1176	0.8235
De'Aaron Fox	1	0.486	0.1306	0.2059	0.7353
	2	0.4483	0.1501	0.1765	0.8529
	3	0.5214	0.1774	0.1471	0.8235
	4	0.4255	0.1302	0.0882	0.7059
Frank Mason	1	0.3846	0.0317	0.0385	0.1154
	2	0.4667	0.0388	0.0769	0.3077
	3	0.3182	0.0279	0.0	0.3462
	4	0.3958	0.0665	0.0385	0.4615
Harry Giles III	1	0.35	0.0244	0.2381	0.381
	2	0.5714	0.0272	0.0476	0.2857
	3	0.3214	0.0355	0.381	0.619
	4	0.5682	0.0609	0.381	0.6667
Iman Shumpert	1	0.3735	0.1013	0.037	0.5556
	2	0.5	0.0569	0.0741	0.7037
	3	0.3651	0.0798	0.1852	0.9259
	4	0.6842	0.0263	0.1481	0.5556
Justin Jackson	1	0.4889	0.0549	0.125	0.3438
	2	0.2941	0.066	0.125	0.7188
	3	0.4865	0.0469	0.0625	0.375
	4	0.5472	0.0734	0.1875	1.0
Kosta Koufos	1	0.5357	0.0342	0.6842	1.2105
	2	0.5789	0.0246	0.3158	0.9474
	3	0.2273	0.0279	0.5263	1.1579
	4	0.5833	0.0166	0.2105	0.7368
Marvin Bagley III	1	0.6	0.0672	0.4231	0.9231
	2	0.5246	0.0789	0.5769	0.8462
	3	0.6	0.057	0.5769	0.6154
	4	0.4583	0.0997	0.8462	1.3077
Nemanja Bjelica	1	0.4118	0.1038	0.4118	1.4118
	2	0.5405	0.0957	0.4412	1.2059
	3	0.5625	0.1014	0.4706	1.2941
	4	0.5526	0.0526	0.1765	0.6176
Skal Labissiere	1	0.0	0.0012	0.0	0.0
	2	0.25	0.0052	0.0	0.0
	3	0.75	0.0051	0.0	0.5556
	4	0.4118	0.0235	0.2222	1.3333
Troy Williams	1	0.25	0.0147	0.1667	0.2778
	2	0.32	0.0323	0.2222	0.3889
	3	0.5625	0.0203	0.0556	0.7222
	4	0.5882	0.0471	0.0556	1.1667
Willie Cauley-Stein	1	0.6075	0.1306	0.4545	1.5152
	2	0.4815	0.1397	0.5758	1.8182
	3	0.4946	0.1179	0.697	1.8788
	4	0.6029	0.0942	0.6364	1.4545
Yogi Ferrell	1	0.2941	0.0208	0.0	0.3043
	2	0.3902	0.053	0.0435	0.3913
	3	0.5625	0.0203	0.0435	0.2609
	4	0.4	0.0554	0.0	0.4783

Close game tendencies (shots in the last 5min of games that ended within 5 pts or in overtime)

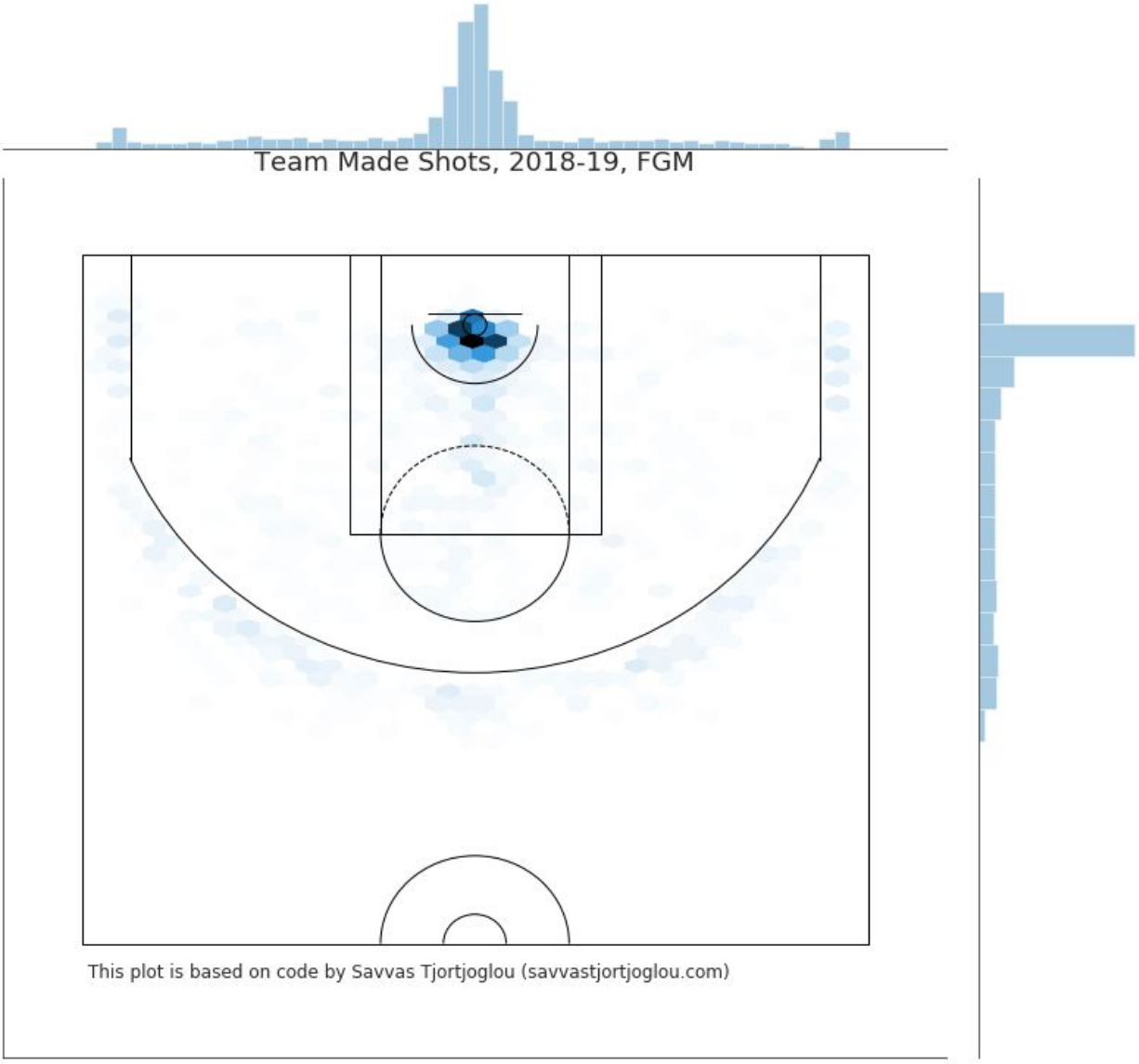
PLAYER_NAME	FG %	% of team-shots taken
Bogdan Bogdanovic	0.4286	0.0722
Buddy Hield	0.4333	0.3093
De'Aaron Fox	0.3478	0.2371
Iman Shumpert	0.5	0.0206
Justin Jackson	0.1667	0.0619
Kosta Koufos	0.0	0.0206
Marvin Bagley III	0.5	0.0619
Nemanja Bjelica	0.75	0.0412
Willie Cauley-Stein	0.5625	0.1649
Yogi Ferrell	0.0	0.0103

Top 2 players in crunch time shot charts:





Overall Team Shooting:



Appendix B - Code - .ipynb file submitted as well

```
#NBA advanced scouting report
#By: Ron Kinel, Kenneth Liu, Zihao Niu, Derek Yan
#Use the below chunk to select desired team and game number and run rest of report

Desired_Team = 'LAL' #Team Abbriviation Here
Game_number = 35 #Game number here
#run entire report
#note: the bash chunk should work since; it is using the box data from assignment 2

#installing NBA api to access team schedules and player rebounds
pip install nba_api

#getting lakers sched from NBA API
%matplotlib inline

import pandas as pd
import numpy as np
import json

from nba_api.stats.static import teams

nba_teams = teams.get_teams()
# Select the dictionary for the desired team, which contains their team ID
Lakers = [team for team in nba_teams if team['abbreviation'] == Desired_Team][0]
Lakers_id = Lakers['id']
from nba_api.stats.endpoints import leaguegamefinder

# Query for games where the selected team is playing were playing
gamefinder = leaguegamefinder.LeagueGameFinder(team_id_nullable=Lakers_id)
# The first DataFrame of those returned is what we want.
games = gamefinder.get_data_frames()[0]
games_season = games[games['SEASON_ID'] == '22018']
games_reg = (pd.to_datetime(games_season.GAME_DATE) > pd.to_datetime('2018-10-01'))
games_season_reg = games_season[games_reg]
lakers_sched = games_season_reg.MATCHUP.str[-3:]
lakers_sched_flip = lakers_sched[::-1]

#All shots from folder
%%bash
```

```
wget -nc -P /home/jovyan/Fall2019/assignments/assignment2/Data
https://ucsb.box.com/shared/static/940qiuxyp798gv4nx8iwvazu9qqjs37e.zip
unzip -o
/home/jovyan/Fall2019/assignments/assignment2/Data/940qiuxyp798gv4nx8iwvazu9qqjs37e.zi
p -d /home/jovyan/Fall2019/assignments/assignment2/Data
unlink
/home/jovyan/Fall2019/assignments/assignment2/Data/shotchartdetail\?PlayerID\=2594\&Playe
rPosition\=\&Season\=2018-19\&ContextMeasure\=FGA\&DateFrom\=\&DateTo\=\&GameID\=\
&GameSegment\=\&LastNGames\=0\&LeagueID\=00\&Location\=\&Month\=0\&OpponentTeam
ID\=0\&Outcome\=\&Period\=0\&Position\=\&RookieYear\=\&SeasonSegment\=\&Se.1
unlink
/home/jovyan/Fall2019/assignments/assignment2/Data/shotchartdetail\?PlayerID\=101181\&Pla
yerPosition\=\&Season\=2018-19\&ContextMeasure\=FGA\&DateFrom\=\&DateTo\=\&GameID\
=\&GameSegment\=\&LastNGames\=0\&LeagueID\=00\&Location\=\&Month\=\0\&OpponentTe
amID\=0\&Outcome\=\&Period\=0\&Position\=\&RookieYear\=\&SeasonSegment\=\&.1

#Allshots taken in 18-19 from HW 2
allshotslist = []

files = !ls -1 /home/jovyan/Fall2019/assignments/assignment2/Data/shotchartdetail*

for f in files:
    edit0_f = f.replace('=', '\=')
    edit1_f = edit0_f.replace('&', '\&')
    edit2_f = edit1_f.replace('?', '\?')
    json_str = !cat {edit2_f}
    json_obj = json.loads(json_str[0])
    h = json_obj['resultSets'][0]['headers']
    d = json_obj['resultSets'][0]['rowSet']
    allshotslist_temp = pd.DataFrame(d, columns=h)
    allshotslist_df = pd.DataFrame(d, columns=h, index = allshotslist_temp.loc[:,'PLAYER_ID'])
    allshotslist.append(allshotslist_df)

allshots = pd.concat(allshotslist)


json_str = !cat
/home/jovyan/Fall2019/assignments/assignment2/Data/commonTeamYears\?LeagueID\=00\&S
eason\=2018-19
json_obj = json.loads(json_str[0])
h = json_obj['resultSets'][0]['headers']
d = json_obj['resultSets'][0]['rowSet']
allteams = pd.DataFrame(d, columns=h)
```

```
#Function to determine OPP team record
def record(i):
   Opponnent = [team for team in nba_teams if team['abbreviation'] ==
lakers_sched_flip.iloc[i-1]][0] ##
   Opponnent_id = Opponnent['id']
   gamefinder_opp = leaguegamefinder.LeagueGameFinder(team_id_nullable=Opponnent_id)
   games_opp = gamefinder_opp.get_data_frames()[0]
   games_season_opp = games_opp[games['SEASON_ID'] == '22018']
   games_season_reg_opp =
games_season_opp[pd.to_datetime(games_season_opp.GAME_DATE) >
pd.to_datetime('2018-10-01')]
   games_before_reg_opp =
games_season_reg_opp[pd.to_datetime(games_season_opp.GAME_DATE) <
pd.to_datetime(games_season_reg.iloc[-i, 5])]
   win = sum(games_before_reg_opp.WL == 'W')
   loss = len(games_before_reg_opp) - win
   return(str(games_opp.TEAM_NAME[0]) + ' (' + str(win) + '-' + str(loss) + ')')

#function to determine if ith game is back-to-back for OPP
def Back_to_back(i):
   Opponnent = [team for team in nba_teams if team['abbreviation'] ==
lakers_sched_flip.iloc[i-1]][0] ##
   Opponnent_id = Opponnent['id']
   gamefinder_opp = leaguegamefinder.LeagueGameFinder(team_id_nullable=Opponnent_id)
   games_opp = gamefinder_opp.get_data_frames()[0]
   games_season_opp = games_opp[games['SEASON_ID'] == '22018']
   games_season_reg_opp =
games_season_opp[pd.to_datetime(games_season_opp.GAME_DATE) >
pd.to_datetime('2018-10-01')]
   games_before_reg_opp =
games_season_reg_opp[pd.to_datetime(games_season_opp.GAME_DATE) <
pd.to_datetime(games_season_reg.iloc[-i, 5])]
   if((pd.to_datetime(games_season_reg.iloc[-i, 5]) -
pd.to_datetime(games_before_reg_opp.iloc[0, 5])).days > 1):
      return('Not 2nd night of back-to-back')
   else:
      return('2nd night of back-to-back')

#gives the OPP relevant shots for the ith game analysis
def relevant(i):
   return(allshots[(allshots.TEAM_ID == int(allteams.TEAM_ID[allteams.ABBREVIATION ==
lakers_sched_flip.iloc[i-1]])) & (allshots.GAME_ID < games_season_reg.iloc[-i, 4])])
```

```
#finds rebounding averages of opponent at ith game
def rebounds(g):
    names = np.unique(FG.index.tolist())


    player_name = []
    for i in names:
        if not i.isdigit():
            player_name.append(i)

    player_ID = []
    for i in player_name:
        ID = np.unique(allshots[allshots['PLAYER_NAME'] == i]['PLAYER_ID'])
        player_ID.append(ID)

    from nba_api.stats.endpoints import playerdashboardbygamesplits
    REB = []

    for i in player_ID: #Game number in here
        REB_DATA = playerdashboardbygamesplits.PlayerDashboardByGameSplits(player_id = i,
season = '2018-19', period = 0, date_to_nullable = games_season_reg.iloc[-g,
5]).get_data_frames()
        REB.append(REB_DATA[2][['OREB', 'DREB']])

    REB_data = REB

    for i in range(len(REB_data)):
        REB_data[i]['PLAYER_NAME'] = player_name[i]
        REB_data[i]['PERIOD'] = range(1,len(REB_data[i]) + 1)
        REB_data[i] = REB_data[i].groupby(['PLAYER_NAME', 'PERIOD']).mean()

    games = []
    for i in range(len(player_ID)): #game number here
        average = allshots[(allshots['PLAYER_ID'] == player_ID[i][0]) & (allshots.GAME_ID <
games_season_reg.iloc[-g, 4])]
        length = len(average['GAME_DATE'].unique())
        games.append(length)

    for i in range(len(REB_data)):
        REB_data[i] = REB_data[i]/games[i]

    return(REB_data)
```

```
#Function from assignment two to plot shot charts
def bin_shots(df, bin_edges, density=False, sigma=1):

    """Given data frame of shots, compute a 2d matrix of binned counts is computed

    Args:
        df: data frame of shotchartdetail from nba.com.
            At the minimum, variables named LOCX and LOCY are required.
        bin_edges: bin edge definition: edges in x and edges in y

    Returns:
        binned: counts
        xedges: bin edges in X direction
        yedges: bin edges in Y direction
    """
    import numpy as np
    from scipy import ndimage
    x = list(np.asarray(df.LOC_X))
    y = list(np.asarray(df.LOC_Y))
    hist = np.histogram2d(x, y, bins = bin_edges)
    binned = hist[0]
    xedges = hist[1]
    yedges = hist[2]

    if density:

        # Recompute 'binned' using "gaussian_filter"
        binned = ndimage.filters.gaussian_filter(binned , sigma = sigma)

        # Normalize the histogram to be a "density", e.g. mass across all bins sums to 1.
        binned /= np.sum(binned)

    return(binned, xedges, yedges)

xedges = np.linspace(start=-300, stop=300, num=151)
yedges = np.linspace(start=-48, stop=372, num=106)
bin_edges = (xedges, yedges)

def plot_shotchart(binned_counts, xedges, yedges, ax=None, use_log=False, cmap = 'Reds'):

    """Plots 2d heatmap from vectorized heatmap counts
```

```
    Args:
        hist_counts: vectorized output of numpy.histogram2d
        xedges, yedges: bin edges in arrays
        ax: figure axes [None]
        use_log: will convert count x to log(x+1) to increase visibility [False]
        cmap: Set the color map https://matplotlib.org/examples/color/colormaps_reference.html
    Returns:
        ax: axes with plot
    """

    import numpy as np
    import matplotlib.pyplot as plt

    ## number of x and y bins.
    nx = xedges.size - 1
    ny = yedges.size - 1

    X, Y = np.meshgrid(xedges, yedges)

    if use_log:
        counts = np.log(binned_counts + 1)

    if ax is None:
        fig, ax = plt.subplots(1,1)

    ax.pcolormesh(X, Y, binned_counts.T, cmap=cmap)
    ax.set_aspect('equal')

    draw_court(ax)

    return(ax)

def draw_court(ax=None, color='black', lw=1, outer_lines=False):

    from matplotlib.patches import Circle, Rectangle, Arc
    from matplotlib.pyplot import gca

    # If an axes object isn't provided to plot onto, just get current one
    if ax is None:
        ax = gca()

    # Create the various parts of an NBA basketball court
```

```
# Create the basketball hoop
# Diameter of a hoop is 18" so it has a radius of 9", which is a value
# 7.5 in our coordinate system
hoop = Circle((0, 0), radius=7.5, linewidth=lw, color=color, fill=False)

# Create backboard
backboard = Rectangle((-30, -7.5), 60, 0, linewidth=lw, color=color)

# The paint
# Create the outer box 0f the paint, width=16ft, height=19ft
outer_box = Rectangle((-80, -47.5), 160, 190, linewidth=lw, color=color,
                      fill=False)
# Create the inner box of the paint, widt=12ft, height=19ft
inner_box = Rectangle((-60, -47.5), 120, 190, linewidth=lw, color=color,
                      fill=False)

# Create free throw top arc
top_free_throw = Arc((0, 142.5), 120, 120, theta1=0, theta2=180,
                     linewidth=lw, color=color, fill=False)
# Create free throw bottom arc
bottom_free_throw = Arc((0, 142.5), 120, 120, theta1=180, theta2=0,
                        linewidth=lw, color=color, linestyle='dashed')
# Restricted Zone, it is an arc with 4ft radius from center of the hoop
restricted = Arc((0, 0), 80, 80, theta1=0, theta2=180, linewidth=lw,
                 color=color)

# Three point line
# Create the side 3pt lines, they are 14ft long before they begin to arc
corner_three_a = Rectangle((-219, -47.5), 0, 140, linewidth=lw,
                           color=color)
corner_three_b = Rectangle((219, -47.5), 0, 140, linewidth=lw, color=color)
# 3pt arc - center of arc will be the hoop, arc is 23'9" away from hoop
# I just played around with the theta values until they lined up with the
# threes
three_arc = Arc((0, 0), 475, 475, theta1=22.5, theta2=157.5, linewidth=lw,
                color=color)

# Center Court
center_outer_arc = Arc((0, 422.5), 120, 120, theta1=180, theta2=0,
                       linewidth=lw, color=color)
center_inner_arc = Arc((0, 422.5), 40, 40, theta1=180, theta2=0,
                       linewidth=lw, color=color)
```

```
    # List of the court elements to be plotted onto the axes
    court_elements = [hoop, backboard, outer_box, inner_box, top_free_throw,
                      bottom_free_throw, restricted, corner_three_a,
                      corner_three_b, three_arc, center_outer_arc,
                      center_inner_arc]

    if outer_lines:
        # Draw the half court line, baseline and side out bound lines
        outer_lines = Rectangle((-250, -47.5), 500, 470, linewidth=lw,
                                color=color, fill=False)
        court_elements.append(outer_lines)

    # Add the court elements onto the axes
    for element in court_elements:
        ax.add_patch(element)

    return ax

import matplotlib.pyplot as plt

#Function to create nice table
import six

def render_mpl_table(data, col_width=3.0, row_height=0.625, font_size=14,
                     header_color='#40466e', row_colors=['#f1f1f2', 'w'], edge_color='w',
                     bbox=[0, 0, 1, 1], header_columns=0,
                     ax=None, **kwargs):
    if ax is None:
        size = (np.array(data.shape[::-1]) + np.array([0, 1])) * np.array([col_width, row_height])
        fig, ax = plt.subplots(figsize=size)
        ax.axis('off')

    mpl_table = ax.table(cellText=data.values, bbox=bbox, colLabels=data.columns, **kwargs)

    mpl_table.auto_set_font_size(False)
    mpl_table.set_fontsize(font_size)

    for k, cell in  six.iteritems(mpl_table._cells):
        cell.set_edgecolor(edge_color)
        if k[0] == 0 or k[1] < header_columns:
            cell.set_text_props(weight='bold', color='w')
            cell.set_facecolor(header_color)
        else:
```

```
            cell.set_facecolor(row_colors[k[0]%len(row_colors) ])
    return ax

#all relevant close game shots
def close_game(i):
    Opponnent = [team for team in nba_teams if team['abbreviation'] ==
lakers_sched_flip.iloc[i-1]][0] ##
    Opponnent_id = Opponnent['id']
    gamefinder_opp = leaguegamefinder.LeagueGameFinder(team_id_nullable=Opponnent_id)
    games_opp = gamefinder_opp.get_data_frames()[0]
    games_season_opp = games_opp[games['SEASON_ID'] == '22018']
    games_season_reg_opp =
games_season_opp[pd.to_datetime(games_season_opp.GAME_DATE) >
pd.to_datetime('2018-10-01')]
    games_before_reg_opp =
games_season_reg_opp[pd.to_datetime(games_season_opp.GAME_DATE) <
pd.to_datetime(games_season_reg.iloc[-i, 5])]
    games_before_reg_opp.GAME_ID[abs(games_before_reg_opp.PLUS_MINUS) < 6] #close
game
    Close_shots = (allshots[(allshots.TEAM_ID ==
int(allteams.TEAM_ID[allteams.ABBREVIATION == lakers_sched_flip.iloc[i-1]])) &
(allshots.GAME_ID.isin(games_before_reg_opp.GAME_ID[abs(games_before_reg_opp.PLUS_
MINUS) < 6]))])
    Close_shots_4 = Close_shots[(Close_shots.PERIOD == 4) &
(Close_shots.MINUTES_REMAINING < 6) ] #only last 5min of 4th
    Close_shots_4 = Close_shots_4.append(allshots[(((allshots.PERIOD) > 4) &
(allshots.TEAM_ID == int(allteams.TEAM_ID[allteams.ABBREVIATION ==
lakers_sched_flip.iloc[i-1]])) & ((allshots.GAME_ID < games_season_reg.iloc[-i, 4])))  ] ) #5th+
qtr (OT)
    return(Close_shots_4)

#begin creating players table
relevant_shots =  relevant(Game_number)
relevant_shots.SHOT_MADE_FLAG = pd.to_numeric(relevant_shots.SHOT_MADE_FLAG)
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
FG = relevant_shots.groupby(['PLAYER_NAME', 'PERIOD']).mean()
PP = (relevant_shots.groupby(['PLAYER_NAME',
'PERIOD']).size()/relevant_shots.groupby(['PERIOD']).size())
FG['proportion']= PP.values

#pulling rebound
REB_data = rebounds(Game_number)
```

```
REB_DF = pd.DataFrame()
for i in range(len(REB_data)):
   REB_DF = REB_DF.append(REB_data[i])

#adding rebounds to table
FG['OREB/period'] = REB_DF.OREB
FG['DREB/period'] = REB_DF.DREB
FG = FG.rename(columns={"SHOT_MADE_FLAG": "FG %", "proportion": "% of team-shots
taken in qtr"})

FGP = FG.round(4).reset_index()
FGP.ix[FGP.duplicated('PLAYER_NAME') , 'PLAYER_NAME'] = ''

record(Game_number)
Back_to_back(Game_number)
render_mpl_table(FGP, header_columns=0, col_width=4.0, font_size=16)

#Plotting top 2 shooters heatmap per qtr
eff = FG
eff["efficiency"] = round(eff["FG %"]*eff["% of team-shots taken in qtr"],4)
sorted_eff = eff.sort_values(["PERIOD","efficiency"], ascending=[True, False])

top2 = sorted_eff.groupby("PERIOD")["efficiency"].nlargest(2)
top2_index = top2.index.get_level_values(level=1)

x = 0
fig, ax = plt.subplots(4,2, figsize = (20,30))

for i in range(1,9):
   if i%2 == 1:
      x = x + 1

   subset = relevant_shots[(relevant_shots["PLAYER_NAME"] == top2_index[i-1]) &
(relevant_shots["SHOT_MADE_FLAG"] == 1) & (relevant_shots["PERIOD"] == x) ]
   subset0, xe, ye, = bin_shots(subset, bin_edges, density=True, sigma=5)
   plot_shotchart(subset0, xe, ye, ax=ax[x-1, (i-1)%2])
   ax[x-1, (i%2) - 1].set_title(top2_index[i-1] + " ( " + "Period: %i" %x + " )",  fontsize=16)

Close_shots_4 = close_game(Game_number)
Close_shots_4.SHOT_MADE_FLAG = pd.to_numeric(Close_shots_4.SHOT_MADE_FLAG)
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
Close_game = Close_shots_4.groupby(['PLAYER_NAME']).mean()
```

```
PP_close = (Close_shots_4.groupby(['PLAYER_NAME']).size())/len(Close_shots_4)
Close_game['% of team-shots taken']= PP_close.values
Close_game = Close_game.rename(columns={"SHOT_MADE_FLAG": "FG %"})
render_mpl_table(Close_game.round(4).reset_index(), header_columns=0, col_width=4.0)

eff2 = Close_game
eff2["efficiency"] = round(eff2["FG %"]*eff2['% of team-shots taken'],4)
sorted_eff2 = eff2.sort_values(["efficiency"], ascending=[True])

top2 = sorted_eff2["efficiency"].nlargest(2)

top2_index = top2.index.get_level_values(level=0)


x = 1

fig, ax = plt.subplots(1,2, figsize = (20,30))

for i in range(1,3):


   subset = Close_shots_4[(Close_shots_4["PLAYER_NAME"] == top2_index[i-1]) &
(Close_shots_4["SHOT_MADE_FLAG"] == 1)]
   subset0, xe, ye, = bin_shots(subset, bin_edges, density=True, sigma=5) #this line fails
   plot_shotchart(subset0, xe, ye, ax=ax[(i-1)])
   ax[(i-1)].set_title(top2_index[i-1] + ' (close game shooting chart)',  fontsize=16)

################# END CHUNK
```