Pascal Siakam's Improvement Year over Year

In this writeup, we will go over the script that analyzes Pascal Siakam's Year over Year performance.

In this 2019-2020 season, Pascal has played a total of 27 games so far. This script will reflect that. We will compare his current per game averages of the first 27 games played compared to the first 27 games played in his previous seasons in the NBA. Pascal has been playing in the NBA for a total of four seasons now, including this current one. The seasons are as follows:

2016 – 2017 season (debut)
2017 – 2018 season
2018 – 2019 season (championship)
2019 – 2020 season (current season)

1. Data Importing

We will be taking the stats from ESPN's game logs and we will be scraping the values using Panda's built in pd.read_html() method. The code to import the season stats is as follows:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# 2020 Season
# Importing game log data from 2019-2020 season for Paskal Siakam
statsbygame2020 = pd.read_html("https://www.espn.com/nba/player/gamelog/_/id/3149673/pascal-
siakam")

data2020 = pd.DataFrame(columns = statsbygame2020[0].columns)

for i in range(len(statsbygame2020) - 3):
    df1 = statsbygame2020[i]
    df1 = df1.iloc[:-1]
    data2020 = data2020.append(df1)

data2020.reset_index(drop = True, inplace = True)
```

pd.read_html() imports the tables on the directed page into a Python list. The list objects themselves are Pandas dataframe objects. We store that list in a created variable, statsbygame2020. There are other methods available as well to scrape tables from html pages, such as the beautifulsoup library, however when pd.read_html() works it is very convenient. We then iterate through this list of dataframes to take only the tables we need. We will be working only with regular season games in this writeup however this can also be easily applied for playoff games by filtering through the list differently. The dataframes of interest are stored in the data2020 dataframe by appending.

This process is now repeated for the remaining 3 seasons. Unfortunately we cannot iterate through these urls as the length of each list generated through pd.read_html() will be different. For example, if the Raptors made the 2nd round of playoffs that season, it will add one more table to ESPN's game log web page. Specifically, we cannot reuse len(statsbygame2020 – 3), we will have to subtract a different integer each time that is manually determined.

```
# 2019 Season
# Importing game log data from 2018-2019 season for Paskal Siakam
statsbygame2019 =
pd.read_html("https://www.espn.com/nba/player/gamelog/_/id/3149673/type/nba/year/2019")
statsbygame2019 = statsbygame2019[5:12]

data2019 = pd.DataFrame(columns = statsbygame2019[0].columns)

# Appending our empty dataframe with relevent rows (last row contains monthly averages)
for i in range(len(statsbygame2019)):
    df2 = statsbygame2019[i]
    df2 = df2.iloc[:-1]
    data2019 = data2019.append(df2)

# Reversing out dataframe so that games are in ascending order (October to April)
data2019 = data2019[::-1]

# Resetting our index
data2019.reset_index(drop = True, inplace = True)



# 2018 Season
statsbygame2018 =
pd.read_html("https://www.espn.com/nba/player/gamelog/_/id/3149673/type/nba/year/2018")
statsbygame2018 = statsbygame2018[3:10]

data2018 = pd.DataFrame(columns = statsbygame2018[5].columns)

# Appending our empty dataframe with relevent rows (last row contains monthly averages)
for i in range(len(statsbygame2018)):
    df3 = statsbygame2018[i]
    df3 = df3.iloc[:-1]
    data2018 = data2018.append(df3)

# Reversing out dataframe so that games are in ascending order (October to April)
data2018 = data2018[::-1]

# Resetting our index
data2018.reset_index(drop = True, inplace = True)



# 2017 Season
statsbygame2017 =
pd.read_html("https://www.espn.com/nba/player/gamelog/_/id/3149673/type/nba/year/2017")
statsbygame2017 = statsbygame2017[2:9]

data2017 = pd.DataFrame(columns = statsbygame2017[5].columns)

# Appending our empty dataframe with relevent rowslast row contains monthly averages)
for i in range(len(statsbygame2017)):
    df4 = statsbygame2017[i]
    df4 = df4.iloc[:-1]
    data2017 = data2017.append(df4)

# Reversing out dataframe so that games are in ascending order (October to April)
data2017 = data2017[::-1]

# Resetting our index
data2017.reset_index(drop = True, inplace = True)
```

In [13]:
```
data2020.head()
```

Out[13]:

	Date	OPP	Result	MIN	FG	FG%	3PT	3P%	FT	FT%	REB	A$T	BLK	$TL	PF	TO	PT$
0	Wed 12/18	@DET	W112-99	35.0	10-23	43.5	6-11	54.5	0-0	0.0	5.0	3.0	4.0	1.0	5.0	2.0	26.0
1	Mon 12/16	vsCLE	W133-113	36.0	13-24	54.2	5-8	62.5	2-2	100.0	4.0	4.0	0.0	0.0	2.0	2.0	33.0
2	Sat 12/14	vsBKN	W110-102	35.0	10-26	38.5	3-5	60.0	7-10	70.0	10.0	5.0	3.0	3.0	4.0	2.0	30.0
3	Wed 12/11	vsLAC	L112-92	35.0	9-20	45.0	1-4	25.0	5-5	100.0	4.0	3.0	3.0	1.0	4.0	3.0	24.0
4	Mon 12/9	@CHI	W93-92	39.0	7-18	38.9	2-6	33.3	6-8	75.0	6.0	1.0	1.0	1.0	0.0	2.0	22.0

In [14]:
```
data2019.head()
```

Out[14]:

	Date	OPP	Result	MIN	FG	FG%	3PT	3P%	FT	FT%	REB	A$T	BLK	$TL	PF	TO	PT$
0	Wed 10/17	vsCLE	W116-104	20.0	6-8	75.0	1-1	100.0	0-0	0.0	2.0	2.0	1.0	0.0	2.0	0.0	13.0
1	Fri 10/19	vsBOS	W113-101	18.0	2-6	33.3	0-2	0.0	0-0	0.0	5.0	2.0	0.0	0.0	1.0	2.0	4.0
2	Sat 10/20	@WSH	W117-113	26.0	4-8	50.0	1-1	100.0	1-3	33.3	10.0	2.0	1.0	1.0	6.0	2.0	10.0
3	Mon 10/22	vsCHA	W127-106	20.0	2-4	50.0	0-1	0.0	0-0	0.0	5.0	2.0	0.0	0.0	3.0	2.0	4.0
4	Wed 10/24	vsMIN	W112-105	20.0	4-6	66.7	1-1	100.0	0-1	0.0	4.0	3.0	2.0	1.0	3.0	0.0	9.0

In [15]:
```
data2018.head()
```

Out[15]:

	Date	OPP	Result	MIN	FG	FG%	3PT	3P%	FT	FT%	REB	A$T	BLK	$TL	PF	TO	PT$
0	Thu 10/19	vsCHI	W117-100	5.0	0-0	0.0	0-0	0.0	0-0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1	Sat 10/21	vsPHI	W128-94	8.0	1-5	20.0	0-2	0.0	2-2	100.0	6.0	2.0	0.0	0.0	1.0	0.0	4.0
2	Wed 10/25	@GS	L117-112	21.0	9-12	75.0	2-3	66.7	0-0	0.0	2.0	2.0	0.0	0.0	1.0	1.0	20.0
3	Fri 10/27	@LAL	W101-92	25.0	8-10	80.0	1-3	33.3	1-2	50.0	5.0	1.0	0.0	2.0	4.0	0.0	18.0
4	Mon 10/30	@POR	W99-85	26.0	3-6	50.0	0-2	0.0	0-0	0.0	5.0	1.0	1.0	2.0	1.0	0.0	6.0

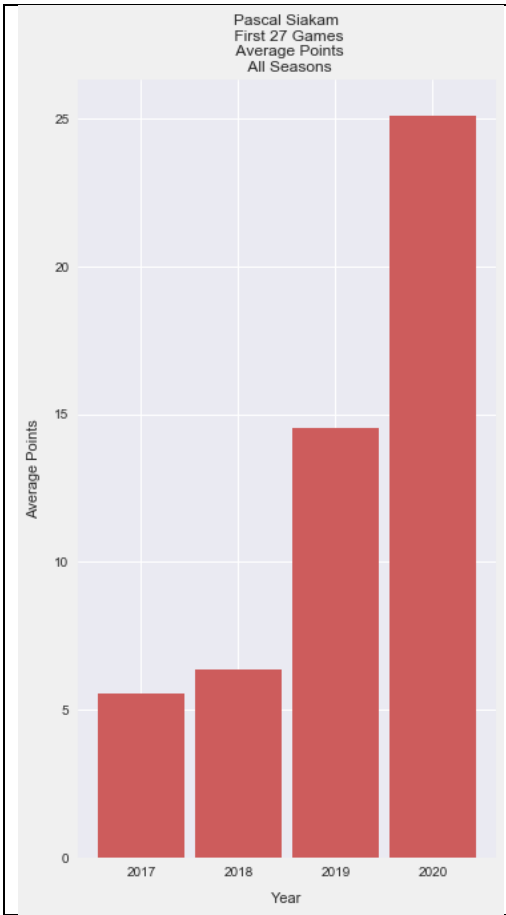
In [16]:
```
data2017.head()
```

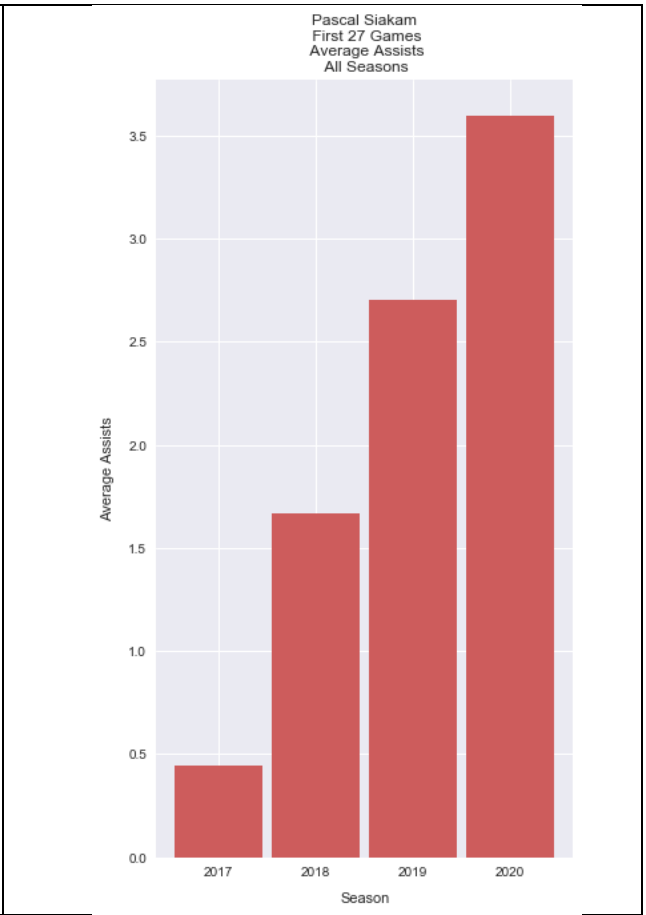
Out[16]:

	Date	OPP	Result	MIN	FG	FG%	3PT	3P%	FT	FT%	REB	A$T	BLK	$TL	PF	TO	PT$
0	Wed 10/26	vsDET	W109-91	22.0	2-2	100.0	0-0	0.0	0-0	0.0	9.0	0.0	0.0	1.0	2.0	0.0	4.0
1	Fri 10/28	vsCLE	L94-91	15.0	2-4	50.0	0-0	0.0	0-0	0.0	3.0	0.0	0.0	1.0	3.0	2.0	4.0
2	Mon 10/31	vsDEN	W105-102	15.0	1-4	25.0	0-0	0.0	0-0	0.0	4.0	0.0	0.0	0.0	3.0	0.0	2.0
3	Wed 11/2	@WSH	W113-103	18.0	2-4	50.0	0-0	0.0	0-0	0.0	3.0	0.0	0.0	1.0	3.0	1.0	4.0
4	Fri 11/4	vsMIA	W96-87	16.0	4-6	66.7	0-0	0.0	0-0	0.0	3.0	0.0	0.0	0.0	0.0	1.0	8.0

Above is what each of these dataframes look like in Jupyter Notebook. Now we can move on to calculating some averages and visualizing the plots.

2. Data Visualization



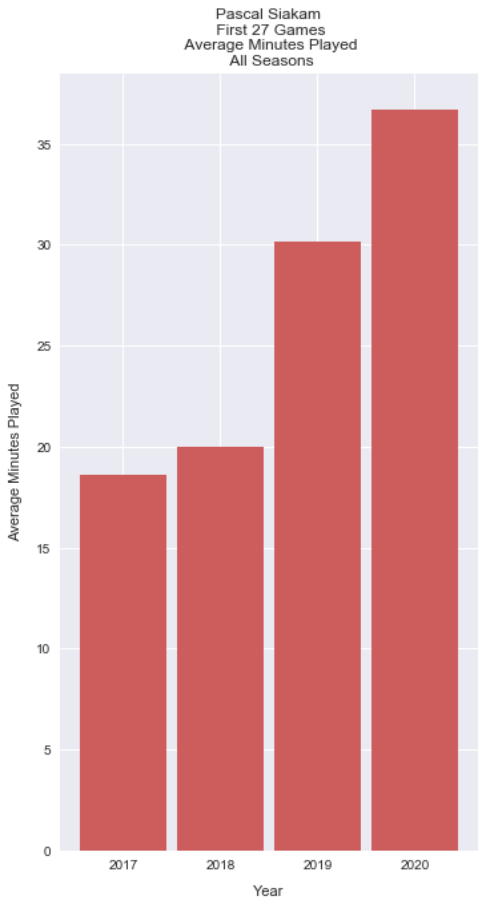


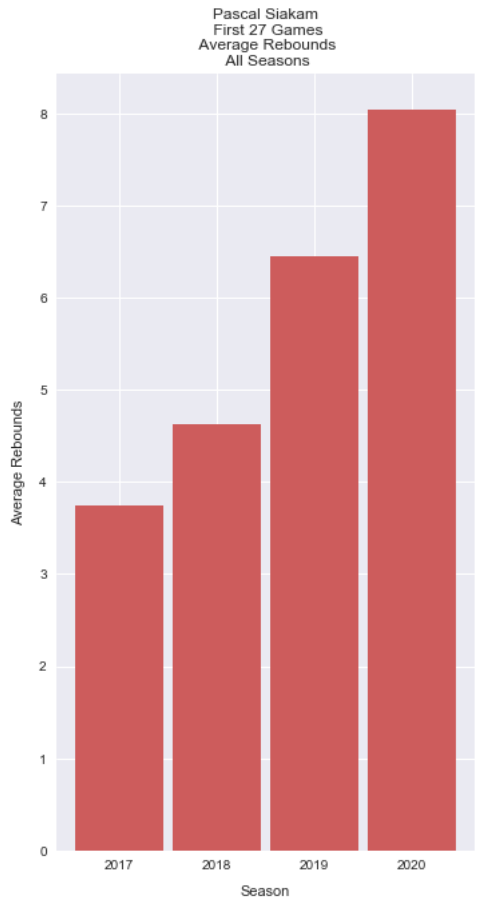

```
# Average Points in Same NUmber of Games in
2019 Season
data2019.iloc[:len(data2020)]['PTS'].mean()
# Average Points in 2020 Season
data2020['PTS'].mean()
# Creating a list to plot the average points
pointscompare =
[data2017.iloc[:len(data2020)]['PTS'].mean(),
data2018.iloc[:len(data2020)]['PTS'].mean(),
data2019.iloc[:len(data2020)]['PTS'].mean(),
data2020['PTS'].mean()]

plt.figure(figsize = (5, 10))
plt.style.use("seaborn")
plt.bar(x = seasons, height = pointscompare,
width = 0.9, color = 'indianred')
plt.title('%s\n First %d Games\n Average
Points\n All Seasons' % (player,
len(data2020)))
plt.xlabel('Year', labelpad = 10)
plt.ylabel('Average Points', labelpad = 10)
```

```
# Assists
assistscompare =
[data2017.iloc[:len(data2020)]['AST'].mean(),
data2018.iloc[:len(data2020)]['AST'].mean(),
data2019.iloc[:len(data2020)]['AST'].mean(),
data2020['AST'].mean()]

plt.figure(figsize = (5, 10))
plt.style.use("seaborn")
plt.bar(x = seasons, height = assistscompare,
width = 0.9, color = 'indianred')
plt.title('%s\n First %d Games\n Average
Assists\n All Seasons' % (player,
len(data2020)))
plt.xlabel('Season', labelpad = 10)
plt.ylabel('Average Assists', labelpad = 10)
```



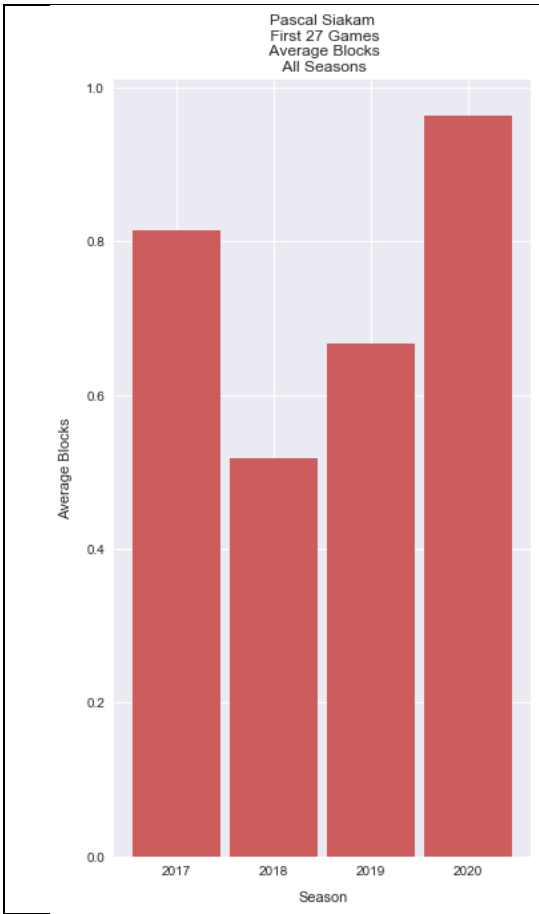


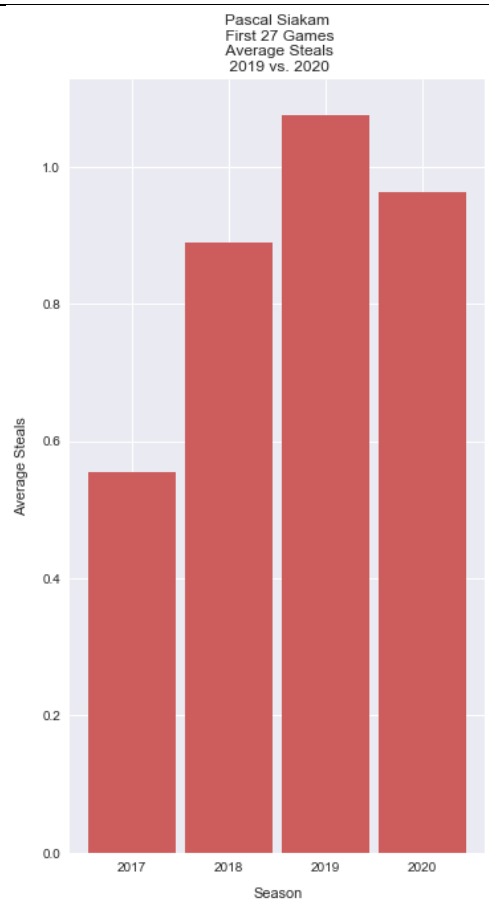

```
# Using the same logic, we can compare the
minutes played
minutescompare =
[data2017.iloc[:len(data2020)]['MIN'].mean(),
data2018.iloc[:len(data2020)]['MIN'].mean(),
data2019.iloc[:len(data2020)]['MIN'].mean(),
data2020['MIN'].mean()]

plt.figure(figsize = (5, 10))
plt.style.use("seaborn")
plt.bar(x = seasons, height = minutescompare,
width = 0.9, color = 'indianred')
plt.title('%s\n First %d Games\n Average
Minutes Played\n All Seasons' % (player,
len(data2020)))
plt.xlabel('Year', labelpad = 10)
plt.ylabel('Average Minutes Played', labelpad =
10)
```

```
reboundscompare =
[data2017.iloc[:len(data2020)]['REB'].mean(),
data2018.iloc[:len(data2020)]['REB'].mean(),
data2019.iloc[:len(data2020)]['REB'].mean(),
data2020['REB'].mean()]

plt.figure(figsize = (5, 10))
plt.style.use("seaborn")
plt.bar(x = seasons, height = reboundscompare,
width = 0.9, color = 'indianred')
plt.title('%s\n First %d Games\n Average
Rebounds\n All Seasons' % (player,
len(data2020)))
plt.xlabel('Season', labelpad = 10)
plt.ylabel('Average Rebounds', labelpad = 10)
```






```
# Blocks
blockscompare =
[data2017.iloc[:len(data2020)]['BLK'].mean(),
data2018.iloc[:len(data2020)]['BLK'].mean(),
data2019.iloc[:len(data2020)]['BLK'].mean(),
data2020['BLK'].mean()]

plt.figure(figsize = (5, 10))
plt.style.use("seaborn")
plt.bar(x = seasons, height = blockscompare,
width = 0.9, color = 'indianred')
plt.title('%s\n First %d Games\n Average
Blocks\n All Seasons' % (player,
len(data2020)))
plt.xlabel('Season', labelpad = 10)
plt.ylabel('Average Blocks', labelpad = 10)
```

```
# Steals
stealscompare =
[data2017.iloc[:len(data2020)]['STL'].mean(),
data2018.iloc[:len(data2020)]['STL'].mean(),
data2019.iloc[:len(data2020)]['STL'].mean(),
data2020['STL'].mean()]

plt.figure(figsize = (5, 10))
plt.style.use("seaborn")
plt.bar(x = seasons, height = stealscompare,
width = 0.9, color = 'indianred')
plt.title('%s\n First %d Games\n Average
Steals\n 2019 vs. 2020' % (player,
len(data2020)))
plt.xlabel('Season', labelpad = 10)
plt.ylabel('Average Steals', labelpad = 10)
```

And just like that we have the average of the first 27 games for each stat in Pascal Siakam's entire NBA career. Notice that out of the 6 statistics we looked at, he had consistent year over year improvement in four of them. Pascal Siakam was the NBA's Most Improved Player in the 2018-2019 season and this is one way that it is showing itself. We can calculate even more statistics, such as the % improvement each year, etc.

We can combine these means into a new dataframe to export as a csv so we can make some slicker visualizations in software such as Tableau.

```
statsdfs = [pointscompare, minutescompare, reboundscompare, assistscompare, blockscompare,
stealscompare]
stats = ['Average Points', 'Average Minutes Played', 'Average Rebounds', 'Average Assists',
'Average Blocks', 'Average Steals']
season = ['2016 - 2017', '2017 - 2018', '2018 - 2019', '2019 - 2020']

export = pd.DataFrame(data = statsdfs, columns = season)
export.index = stats
```

```
In [31]: export.head()
```

Out[31]:

	2016 - 2017	2017 - 2018	2018 - 2019	2019 - 2020
Average Points	5.555556	6.370370	14.518519	25.074074
Average Minutes Played	18.629630	19.962963	30.148148	36.666667
Average Rebounds	3.740741	4.629630	6.444444	8.037037
Average Assists	0.444444	1.666667	2.703704	3.592593
Average Blocks	0.814815	0.518519	0.666667	0.962963

3. Looking for Players with Similar Output

Everyone is in agreeance that Pascal Siakam is good. But how good is he? Let's look for players that produce similar outputs to him. To do this, we will need to import the per game averages for all players in the league. These values were taken from Basketball-Reference.

```
stats2020 = pd.read_csv("G:/Dropbox/Software Development/FILES/SATB/NBA/2020 Season Stats Per
Game 12 23 2019.csv")
```

From here, we can create a loop to check for each individual statistic column for the players that have similar output. The loop will first sort the stats2020 dataframe by the first stat, minutes played. It will then reset the indices and look for the index where Pascal Siakam sits. It will then look at the 5 players that produce above him and the five that produce below him and it will append these players and statistics to a dataframe. After iterating this procedure through every single column, we will have a new dataframe with certain players appearing multiple times. We can then create a groupby object grouping this dataframe by players. By counting the number of times a certain player appears in the dataframe, we know how many times this player produced a statistic close to Pascal Siakam.

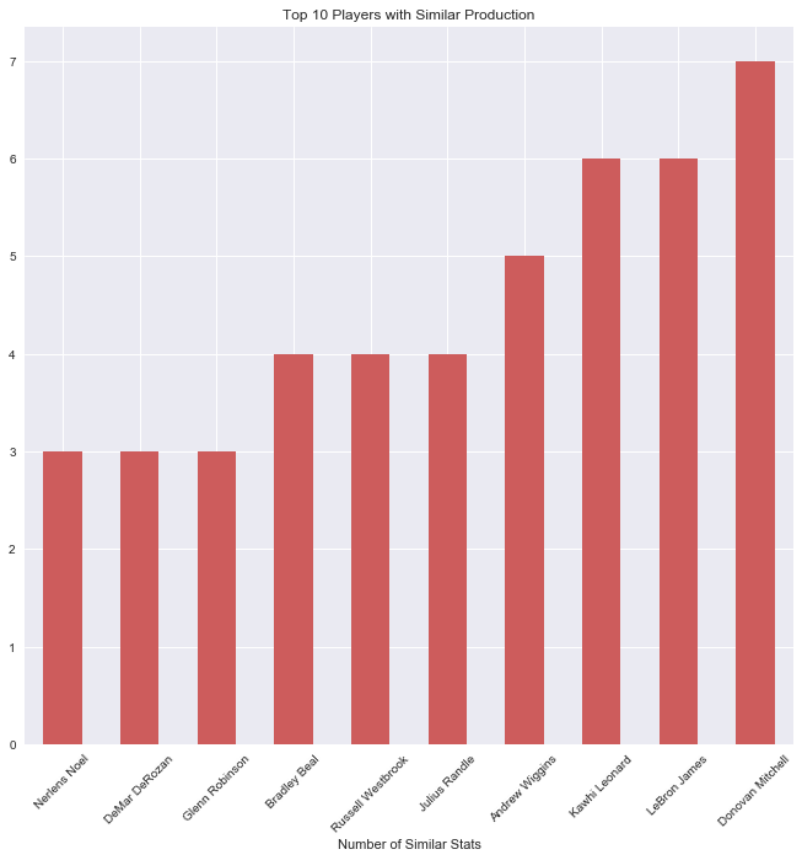
```
statsresult = pd.DataFrame(columns = stats2020.columns)
for stat in stats:
        stats2020.sort_values(stat, ascending = False, inplace = True)
        stats2020.reset_index(drop = True, inplace = True)
        index = stats2020[stats2020['Player'] == 'Pascal Siakam'].index[0]
        statsresult = statsresult.append(stats2020.iloc[index-5:index+6])

groups2020 = statsresult.groupby('Player')
similarstats = groups2020.size().sort_values()

similarstats.tail()
```

```
> Player
> DeMar DeRozan          3
> Glenn Robinson         3
> Bradley Beal           4
> Russell Westbrook      4
> Julius Randle          4
> Andrew Wiggins         5
> Kawhi Leonard          6
> LeBron James           6
> Donovan Mitchell       7
> Pascal Siakam         22
> dtype: int64
```

We now return the top 10 players who produced stats similar to Pascal. Let's plot this.



Not a bad list. We have appearance by Demar DeRozan, Russell Westbrook, Kawhi Leonard and Lebron James. Some big name players. Interestingly, the player that produces the most stats similar to Pascal is Donovan Mitchell.