Using Summary Statistics to Examine the "Hot Hand"

Import useful libraries and the updated shot log data

In [1]:
```
import pandas as pd
import numpy as np
import datetime as dt

Shotlog=pd.read_csv("../../Data/Week 6/Shotlog1.csv")
Player_Stats=pd.read_csv("../../Data/Week 6/Player_Stats1.csv")
Player_Shots=pd.read_csv("../../Data/Week 6/Player_Shots1.csv")
Player_Game=pd.read_csv("../../Data/Week 6/Player_Game1.csv")
Shotlog.head()
```

Out[1]:

	team_previous_shot	player_position	home_game	location_x	opponent_previous_shot	home_team	shot_type	points	away_team	location_y	...	date
0	MISSED	SF	Yes	279.0	SCORED	ATL	Jump Shot	3	WAS	130.0	...	2016-10-27
1	MISSED	SF	Yes	58.0	SCORED	ATL	Cutting Layup Shot	2	WAS	275.0	...	2016-10-27
2	SCORED	SF	Yes	868.0	SCORED	ATL	Jump Shot	3	WAS	475.0	...	2016-10-27
3	SCORED	SF	Yes	691.0	MISSED	ATL	Pullup Jump Shot	3	WAS	100.0	...	2016-10-27
4	MISSED	SF	Yes	691.0	MISSED	ATL	Pullup Jump Shot	2	WAS	181.0	...	2016-10-27

5 rows × 21 columns

Conditional Probability

We can first calculate the conditional probability of making a shot in the current period conditional on making the previous shot.

$$Conditional\ Probability = \frac{Probability\ of\ Making\ Consecutive\ Shots}{Probability\ of\ Making\ Previous\ Shot}$$

We will need to create a variable that indicates a player made consecutive shots.

In [2]:
```
Shotlog['conse_shot_hit'] = np.where((Shotlog['current_shot_hit']==1)&(Shotlog['lag_shot_hit']==1), 1, 0)
Shotlog.head()
```

Out[2]:

	team_previous_shot	player_position	home_game	location_x	opponent_previous_shot	home_team	shot_type	points	away_team	location_y	...	shoot
0	MISSED	SF	Yes	279.0	SCORED	ATL	Jump Shot	3	WAS	130.0	...	Ba
1	MISSED	SF	Yes	58.0	SCORED	ATL	Cutting Layup Shot	2	WAS	275.0	...	Ba
2	SCORED	SF	Yes	868.0	SCORED	ATL	Jump Shot	3	WAS	475.0	...	Ba
3	SCORED	SF	Yes	691.0	MISSED	ATL	Pullup Jump Shot	3	WAS	100.0	...	Ba
4	MISSED	SF	Yes	691.0	MISSED	ATL	Pullup Jump Shot	2	WAS	181.0	...	Ba

5 rows × 22 columns

We can create a player level dataframe. The average of the variable "conse_shot_hit" would be the joint probability of making current and previous shots. We will also calculate the average of "lag_shot_hit" to indicate the probability of making the previous shot.

In [3]:
```
Player_Prob=Shotlog.groupby(['shoot_player'])['conse_shot_hit','lag_shot_hit'].mean()
Player_Prob=Player_Prob.reset_index()
Player_Prob.rename(columns={'lag_shot_hit':'average_lag_hit'}, inplace=True)
Player_Prob.head()
```

Out[3]:

	shoot_player	conse_shot_hit	average_lag_hit
0	A.J. Hammons	0.148148	0.407407
1	Aaron Brooks	0.184874	0.436975
2	Aaron Gordon	0.204082	0.460459
3	Adreian Payne	0.153846	0.435897
4	Al Horford	0.207367	0.469304

Calculate conditional probability for each player

We can calculate the conditional probability by dividing the joint probability by the probability of making the previous shot.

In [4]:
```
Player_Prob['conditional_prob']=Player_Prob['conse_shot_hit']/Player_Prob['average_lag_hit']
Player_Prob.head()
```

Out[4]:

	shoot_player	conse_shot_hit	average_lag_hit	conditional_prob
0	A.J. Hammons	0.148148	0.407407	0.363636
1	Aaron Brooks	0.184874	0.436975	0.423077
2	Aaron Gordon	0.204082	0.460459	0.443213
3	Adreian Payne	0.153846	0.435897	0.352941
4	Al Horford	0.207367	0.469304	0.441860

We can merge the "Player_Prob" data frame with the "Player_Stats" data frame we created earlier to compare the conditional probability and the unconditional probability. If the two probabilities are the same, or almost the same, then we fail to find evidence that the making the current shot depends on making the previous shot.

In [5]:
```
Player_Stats=pd.merge(Player_Prob, Player_Stats, on=['shoot_player'])
Player_Stats.head(10)
```

Out[5]:

	shoot_player	conse_shot_hit	average_lag_hit	conditional_prob	average_hit
0	A.J. Hammons	0.148148	0.407407	0.363636	0.404762
1	Aaron Brooks	0.184874	0.436975	0.423077	0.403333
2	Aaron Gordon	0.204082	0.460459	0.443213	0.454861
3	Adreian Payne	0.153846	0.435897	0.352941	0.425926
4	Al Horford	0.207367	0.469304	0.441860	0.473159
5	Al Jefferson	0.226601	0.517241	0.438095	0.498938
6	Al-Farouq Aminu	0.170370	0.402469	0.423313	0.392704
7	Alan Anderson	0.134615	0.423077	0.318182	0.375000
8	Alan Williams	0.266667	0.524444	0.508475	0.516854
9	Alec Burks	0.159420	0.391304	0.407407	0.399194

Let's first take a quick look at our "Player_Stats" data frame.

In [6]:
```
Player_Stats.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 481 entries, 0 to 480
Data columns (total 5 columns):
shoot_player        481 non-null object
conse_shot_hit      481 non-null float64
average_lag_hit     481 non-null float64
conditional_prob    472 non-null float64
average_hit         481 non-null float64
dtypes: float64(4), object(1)
memory usage: 22.5+ KB
```

Note that when we created the "conditional_prob" variable, some observations may have missing value since the "average_lag_shot" variable may contain zero value. We will delete these observations with missing values in conditional probability.

In [7]:
```
Player_Stats=Player_Stats[pd.notnull(Player_Stats["conditional_prob"])]
```

We can first check which players have the highest conditional probability, i.e., more likely to have hot hand.

Let's sort the data by conditional probability.

In [8]:
```
Player_Stats.sort_values(by=['conditional_prob'], ascending=[False]).head(10)
```

Out[8]:

	shoot_player	conse_shot_hit	average_lag_hit	conditional_prob	average_hit
72	Chinanu Onuaku	0.333333	0.333333	1.000000	0.714286
293	Lucas Nogueira	0.495495	0.657658	0.753425	0.660256
221	Joel Anthony	0.375000	0.500000	0.750000	0.625000
343	Nick Collison	0.428571	0.571429	0.750000	0.608696
36	Axel Toupane	0.500000	0.666667	0.750000	0.555556
402	Salah Mejri	0.475610	0.658537	0.722222	0.642336
464	Udonis Haslem	0.357143	0.500000	0.714286	0.478261
102	DeAndre Jordan	0.493952	0.699597	0.706052	0.714038
398	Rudy Gobert	0.457565	0.653137	0.700565	0.662921
187	Jakob Poeltl	0.394366	0.563380	0.700000	0.582609

Comparing the "conditional_prob" variable and the "average_hit" variable, some players have a slightly higher conditional probability but some also have a lower conditional probability.

We can sort the data by the value of difference between conditional and unconditional probabilities.

In [9]:
```
Player_Stats['diff_prob']=Player_Stats['conditional_prob']-Player_Stats['average_hit']
Player_Stats=pd.merge(Player_Stats, Player_Shots, on=['shoot_player'])
Player_Stats.sort_values(by=['diff_prob'], ascending=[False]).head(10)
```

Out[9]:

	shoot_player	conse_shot_hit	average_lag_hit	conditional_prob	average_hit	diff_prob	shot_count
276	Lamar Patterson	0.090909	0.181818	0.500000	0.200000	0.300000	15
70	Chinanu Onuaku	0.333333	0.333333	1.000000	0.714286	0.285714	7
120	Diamond Stone	0.125000	0.250000	0.500000	0.230769	0.269231	13
455	Udonis Haslem	0.357143	0.500000	0.714286	0.478261	0.236025	23
215	Joe Young	0.217391	0.369565	0.588235	0.361111	0.227124	72
273	Kyle Wiltjer	0.166667	0.333333	0.500000	0.285714	0.214286	14
300	Marcus Georges-Hunt	0.250000	0.500000	0.500000	0.285714	0.214286	7
72	Chris McCullough	0.388889	0.555556	0.700000	0.500000	0.200000	32
35	Axel Toupane	0.500000	0.666667	0.750000	0.555556	0.194444	9
86	Damjan Rudez	0.230769	0.423077	0.545455	0.352273	0.193182	88

Comparing the "conditional_prob" variable and the "average_hit" variable, some players have a slightly higher conditional probability but some also have a lower conditional probability. We can sort the data by the value of the difference between conditional and unconditional probabilities. We can see that Lamar Patterson has the highest difference between the two probabilities, at 30%. But we could also see that the sample size for Patterson is pretty small. For Joe Young and Damjan Rudez, we have about 80 observations and the difference in the probabilities is about 20%.

T-test for statistical significance on the difference

More rigorously, we can use a t-test to test if the players' probability of hitting the goal is statistically significantly different than their conditional probability.

We need to choose a significance level before we perform the test. If the test produces a p-value less than the chosen significance level, then we say that there is a statistically significant difference between the two probabilities; otherwise, we fail to find evidence to support that the two probabilities are statistically significantly different from each other.

The most commonly used significance level is 0.05.

To perform a t-test, we need to import a new library, "scipy.stats."

In [10]:
```
import scipy.stats as sp
```

We can use the ttest_ind() function to calculate the test statistics.

In [11]:
```
sp.stats.ttest_ind(Player_Stats['conditional_prob'], Player_Stats['average_hit'])
```

Out[11]:
```
Ttest_indResult(statistic=-1.6259251871488678, pvalue=0.10430003992138248)
```

The first number is the t-statistics and the second number is the p-value.

Note that the p-value for the t test is about 0.10, which is higher than the conventional significance level 0.05. Thus the conditional probability is not statistically significantly different than the average success rate. In other words, in the analysis of conditional probability, we fail to find evidence to support the "hot hand".

Autocorrelation Coefficient

We can calculate the autocorrelation coefficient by calculating the correlation coefficient between the "current_shot_hit" variable and the "lag_shot_hit" variable.

Note: in python, you could use "autocorr(lag=1)" to calculate first order autocorrelation coefficient. This command is not very useful in our case since we want to look at the autocorrelation coefficient within each game. Using the built-in autocorrelation coefficient function in python, we will be treating the last shot from the previous game and the first shot of the subsequent game as a pair.

In [12]:
```
Shotlog['current_shot_hit'].corr(Shotlog['lag_shot_hit'])
```

Out[12]:
```
-0.007323295128030561
```

As we can see, the autocorrelation coefficient is negative and the magnitude is very small and close to zero.

Since some players may have "hot hand", and hence strong correlation between outcomes of adjacent shots, while some may not. We can also calculate autocorrelation coefficient for each player.

In [13]:
```
Shotlog.groupby('shoot_player')[['current_shot_hit','lag_shot_hit']].corr().head(10)
```

Out[13]:

		current_shot_hit	lag_shot_hit
shoot_player			
A.J. Hammons	**current_shot_hit**	1.000000	-0.011562
	lag_shot_hit	-0.011562	1.000000
Aaron Brooks	**current_shot_hit**	1.000000	0.012709
	lag_shot_hit	0.012709	1.000000
Aaron Gordon	**current_shot_hit**	1.000000	-0.036678
	lag_shot_hit	-0.036678	1.000000
Adreian Payne	**current_shot_hit**	1.000000	-0.236082
	lag_shot_hit	-0.236082	1.000000
Al Horford	**current_shot_hit**	1.000000	-0.056836
	lag_shot_hit	-0.056836	1.000000

We may not want to print out a 2 by 2 matrix for every player. We can use the "unstack()" command to reshape the data.

In [14]:
```
Autocorr_Hit=Shotlog.groupby('shoot_player')[['current_shot_hit','lag_shot_hit']].corr().unstack()
Autocorr_Hit.head()
```

Out[14]:

	current_shot_hit		lag_shot_hit	
	current_shot_hit	lag_shot_hit	current_shot_hit	lag_shot_hit
shoot_player				
A.J. Hammons	1.0	-0.011562	-0.011562	1.0
Aaron Brooks	1.0	0.012709	0.012709	1.0
Aaron Gordon	1.0	-0.036678	-0.036678	1.0
Adreian Payne	1.0	-0.236082	-0.236082	1.0
Al Horford	1.0	-0.056836	-0.056836	1.0

Note that now each row represents a single player. But we still have duplicate information in the columns.

We can use the ".iloc" command to select the columns that we need.

- In the iloc[,] command, we first specify the rows we want to select, then the columns, i.e., [rows, columns]
- We want to select all rows, so we will have iloc[:,]
- We only want to select the second column, which is indexed 1 (first column would be indexed 0, etc.)
- So we will use the command iloc[:,1]

Lastly, we will also reset the index so that the player names would become a variable.

In [15]:
```
Autocorr_Hit=Shotlog.groupby('shoot_player')[['current_shot_hit','lag_shot_hit']].corr().unstack().iloc[:,1].reset_index()
Autocorr_Hit.head()
```

Out[15]:

	shoot_player	current_shot_hit lag_shot_hit
0	A.J. Hammons	-0.011562
1	Aaron Brooks	0.012709
2	Aaron Gordon	-0.036678
3	Adreian Payne	-0.236082
4	Al Horford	-0.056836

Notice that we still have two levels of variable names.

We can use the "get_level_values" command to reset the variable name to the first level (index 0).

In [16]:
```
Autocorr_Hit.columns=Autocorr_Hit.columns.get_level_values(0)
Autocorr_Hit.head()
```

Out[16]:

	shoot_player	current_shot_hit

0	A.J. Hammons	-0.011562
1	Aaron Brooks	0.012709
2	Aaron Gordon	-0.036678
3	Adreian Payne	-0.236082
4	Al Horford	-0.056836

Let's rename the variable capturing autocorrelation coefficient.

In [17]:
```
Autocorr_Hit.rename(columns={'current_shot_hit':'autocorr'}, inplace=True)
Autocorr_Hit.head()
```

Out[17]:

	shoot_player	autocorr
0	A.J. Hammons	-0.011562
1	Aaron Brooks	0.012709
2	Aaron Gordon	-0.036678
3	Adreian Payne	-0.236082
4	Al Horford	-0.056836

How informative the autocorrelation coefficient also depends on the number of shots per game for each player. Let's add the number of shots and the number of shots per game to the autocorrelation matrix and sort the data by the size of autocorrelation coefficient.

In [18]:
```
Player_Game_Shot=Player_Game.groupby(["shoot_player"])['shot_per_game'].mean().reset_index(name='avg_shot_game')
Player_Game_Shot.head()
```

Out[18]:

	shoot_player	avg_shot_game
0	A.J. Hammons	2.80000
1	Aaron Brooks	4.83871
2	Aaron Gordon	10.80000
3	Aaron Harrison	1.00000
4	Adreian Payne	3.60000

In [19]:
```
Autocorr_Hit=pd.merge(Autocorr_Hit, Player_Game_Shot, on=['shoot_player'])
Autocorr_Hit.sort_values(by=['autocorr'], ascending=[False]).head(10)
```

Out[19]:

	shoot_player	autocorr	avg_shot_game
279	Kyle Wiltjer	0.632456	1.750000
464	Udonis Haslem	0.428571	2.555556
330	Mike Miller	0.414758	2.300000
282	Lamar Patterson	0.388889	3.750000
122	Diamond Stone	0.333333	2.600000
229	Johnny O'Bryant III	0.301511	3.000000
36	Axel Toupane	0.250000	3.000000
202	Jarrod Uthoff	0.237826	4.750000
220	Joe Young	0.237004	2.769231
187	Jakob Poeltl	0.219336	2.613636

We will merge the Player_Game_Shot dataframe to the Player_Shots dataframe since both dataframes are measured in player level and both contain information on the number of shots.

In [20]:
```
Player_Shots=pd.merge(Player_Shots, Player_Game_Shot, on=['shoot_player'])
Player_Shots.head()
```

Out[20]:

	shoot_player	shot_count	avg_shot_game
0	A.J. Hammons	42	2.80000
1	Aaron Brooks	300	4.83871
2	Aaron Gordon	864	10.80000
3	Aaron Harrison	4	1.00000
4	Adreian Payne	54	3.60000

Save updated data

In [21]:
```
Shotlog.to_csv("../../Data/Week 6/Shotlog2.csv", index=False)
Player_Stats.to_csv("../../Data/Week 6/Player_Stats2.csv", index=False)
Player_Shots.to_csv("../../Data/Week 6/Player_Shots2.csv", index=False)
```

In [ ]:
```

```