Pythagorean Expectation and the NBA

The NBA is the most popular basketball league in the world, and consists of 30 teams playing an 82 game regular season followed by playoffs to determine the champion. In terms of scale, this data looks much more like MLB data than the IPL data we just looked at.

Basketball resembles cricket in one way - the scores are much higher than in baseball. However, the points difference between winning and losing teams tend to be relatively small.

Let's see what we find this time. We follow the same procedure.

In [1]:
```
# Load the packages

import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Load the data and see what it looks like

NBA = pd.read_csv('../../Data/Week 1/NBA_Games.csv')

NBA
```

Out[2]:

	CITY	TEAM_NAME	TEAM_ID	GAME_ID	NICKNAME	STATE	YEAR_FOUNDED	SEASON_ID	TEAM_ABBREVIATION	GAME
0	Oklahoma City	Oklahoma City Thunder	1.610613e+09	1.130000e+07	Thunder	Oklahoma	1967.0	12013.0	OKC	10
1	Chicago	Chicago Bulls	1.610613e+09	1.130000e+07	Bulls	Illinois	1966.0	12013.0	CHI	10
2	Indiana	Indiana Pacers	1.610613e+09	1.130000e+07	Pacers	Indiana	1976.0	12013.0	IND	10
3	New Orleans	New Orleans Pelicans	1.610613e+09	1.130000e+07	Pelicans	Louisiana	2002.0	12013.0	NOP	10
4	Houston	Houston Rockets	1.610613e+09	1.130000e+07	Rockets	Texas	1967.0	12013.0	HOU	10
5	Golden State	Golden State Warriors	1.610613e+09	1.130000e+07	Warriors	California	1946.0	12013.0	GSW	10
6	Los Angeles	Los Angeles Lakers	1.610613e+09	1.130000e+07	Lakers	California	1948.0	12013.0	LAL	10

In [3]:
```
# The data consists of games played between 2013 and 2019. An important difference from the baseball and cricket data
# is that here each game appears in two rows, one for each team. Each pair of rows are mirror images of each other.

# The season is identified by the column SEASON_ID
# Pre-season games have the prefix "1" before the year, regular season games have the prefix "2"
# and postseason games have the prefix "4". We are going to look at the 2018 regular season and therefore
# want games with the prefix "2"
# We can use the command ".describe()" to obtain descriptive statistics for our variables.

NBAR18 = NBA[NBA.SEASON_ID == 22018]
NBAR18.describe()
```

Out[3]:

	TEAM_ID	GAME_ID	YEAR_FOUNDED	SEASON_ID	MIN	PTS	PTSAGN	FGM	FGA	FG_PCT	...	
count	2.560000e+03	2.560000e+03	2560.000000	2560.0	2560.000000	2560.000000	2472.000000	2560.000000	2560.000000	2560.000000	...	256
mean	1.610613e+09	1.279724e+08	1969.629688	22018.0	238.953125	109.191406	109.080097	40.359375	88.205469	0.458005	...	10
std	8.753502e+00	3.841538e+08	16.740380	0.0	12.705895	14.344124	14.392685	5.786935	8.219058	0.054905	...	3
min	1.610613e+09	2.180000e+07	1946.000000	22018.0	197.000000	53.000000	53.000000	17.000000	55.000000	0.262000	...	1
25%	1.610613e+09	2.180033e+07	1949.000000	22018.0	239.000000	100.000000	100.000000	37.000000	83.000000	0.420750	...	8
50%	1.610613e+09	2.180066e+07	1970.000000	22018.0	240.000000	110.000000	110.000000	40.000000	88.000000	0.458000	...	10
75%	1.610613e+09	2.180099e+07	1980.000000	22018.0	241.000000	118.000000	118.000000	44.000000	93.000000	0.494000	...	13
max	1.610613e+09	1.621800e+09	2002.000000	22018.0	341.000000	168.000000	168.000000	61.000000	123.000000	0.649000	...	26

8 rows × 26 columns

In [4]:
```
# We can list all the variable names

print(NBAR18.columns.tolist())
```

```
['CITY', 'TEAM_NAME', 'TEAM_ID', 'GAME_ID', 'NICKNAME', 'STATE', 'YEAR_FOUNDED', 'SEASON_ID', 'TEAM_ABBREVIATION', 'GAME_DAT
E', 'MATCHUP', 'WL', 'MIN', 'PTS', 'PTSAGN', 'FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG3A', 'FG3_PCT', 'FTM', 'FTA', 'FT_PCT', 'ORE
B', 'DREB', 'REB', 'AST', 'STL', 'BLK', 'TOV', 'PF', 'PLUS_MINUS', 'home', 'not paired']
```

In [5]:
```
# Many datasets contain missing variables. Missing variables in a column will usually cause operations to fail.
# The command ".dropna()" will eliminate missing variables.
# Compare the counts of variables below after the .dropna() below to the counts in the cell above.

NBAR18 = NBAR18.dropna()
NBAR18.describe()
```

Out[5]:

	TEAM_ID	GAME_ID	YEAR_FOUNDED	SEASON_ID	MIN	PTS	PTSAGN	FGM	FGA	FG_PCT	...	
count	2.472000e+03	2.472000e+03	2472.000000	2472.0	2472.000000	2472.000000	2472.000000	2472.000000	2472.000000	2472.00000	...	2472.
mean	1.610613e+09	1.281113e+08	1969.664644	22018.0	238.945388	109.080097	109.080097	40.338592	88.141181	0.45810	...	10.
std	8.760706e+00	3.843674e+08	16.744769	0.0	12.720983	14.392685	14.392685	5.803877	8.224533	0.05509	...	3.
min	1.610613e+09	2.180000e+07	1946.000000	22018.0	197.000000	53.000000	53.000000	17.000000	55.000000	0.26200	...	1.
25%	1.610613e+09	2.180033e+07	1949.000000	22018.0	239.000000	100.000000	100.000000	37.000000	83.000000	0.42075	...	8.
50%	1.610613e+09	2.180066e+07	1970.000000	22018.0	240.000000	110.000000	110.000000	40.000000	88.000000	0.45800	...	10.
75%	1.610613e+09	2.180099e+07	1980.000000	22018.0	241.000000	118.000000	118.000000	44.000000	93.000000	0.49400	...	13.
max	1.610613e+09	1.621800e+09	2002.000000	22018.0	341.000000	168.000000	168.000000	61.000000	123.000000	0.64900	...	26.

8 rows × 26 columns

In [6]:
```
# The game result is the column labeled 'WL'. We create a variable which has a value of '1' if the team won, and zero if it l
# This type of variable, where a condition (here winning) is either true (1) or not true (0) is called a "dummy variable".
# We will encounter them frequently.

NBAR18['result'] = np.where(NBAR18['WL']== 'W',1,0)
NBAR18.describe()
```

Out[6]:

	TEAM_ID	GAME_ID	YEAR_FOUNDED	SEASON_ID	MIN	PTS	PTSAGN	FGM	FGA	FG_PCT	...	
count	2.472000e+03	2.472000e+03	2472.000000	2472.0	2472.000000	2472.000000	2472.000000	2472.000000	2472.000000	2472.00000	...	2472.
mean	1.610613e+09	1.281113e+08	1969.664644	22018.0	238.945388	109.080097	109.080097	40.338592	88.141181	0.45810	...	34.
std	8.760706e+00	3.843674e+08	16.744769	0.0	12.720983	14.392685	14.392685	5.803877	8.224533	0.05509	...	5.
min	1.610613e+09	2.180000e+07	1946.000000	22018.0	197.000000	53.000000	53.000000	17.000000	55.000000	0.26200	...	17.
25%	1.610613e+09	2.180033e+07	1949.000000	22018.0	239.000000	100.000000	100.000000	37.000000	83.000000	0.42075	...	30.
50%	1.610613e+09	2.180066e+07	1970.000000	22018.0	240.000000	110.000000	110.000000	40.000000	88.000000	0.45800	...	34.
75%	1.610613e+09	2.180099e+07	1980.000000	22018.0	241.000000	118.000000	118.000000	44.000000	93.000000	0.49400	...	38.
max	1.610613e+09	1.621800e+09	2002.000000	22018.0	341.000000	168.000000	168.000000	61.000000	123.000000	0.64900	...	55.

8 rows × 27 columns

In [7]:
```
# For the Pythagorean Expectation we need only the result, points scored (PTS) and point conceded (PTSAGN).

NBAteams18 = NBAR18.groupby('TEAM_NAME')['result','PTS','PTSAGN'].sum().reset_index()
NBAteams18
```

Out[7]:

	TEAM_NAME	result	PTS	PTSAGN
0	Atlanta Hawks	30	9742.0	10306.0
1	Boston Celtics	53	9489.0	9082.0
2	Brooklyn Nets	42	9375.0	9443.0
3	Charlotte Hornets	42	9290.0	9359.0
4	Chicago Bulls	24	8783.0	9467.0
5	Cleveland Cavaliers	24	8976.0	9697.0
6	Dallas Mavericks	34	8910.0	9006.0
7	Denver Nuggets	54	9003.0	8714.0
8	Detroit Pistons	43	9020.0	9069.0
9	Golden State Warriors	57	9677.0	9201.0
10	Houston Rockets	55	9356.0	8963.0

In [8]:
```
# So now we can create the value for win percentage for each team in the 82 game season, and the Pythagorean Expectation.

NBAteams18['wpc'] = NBAteams18['result']/82
NBAteams18['pyth'] = NBAteams18['PTS']**2/(NBAteams18['PTS']**2 + NBAteams18['PTSAGN']**2)
NBAteams18
```

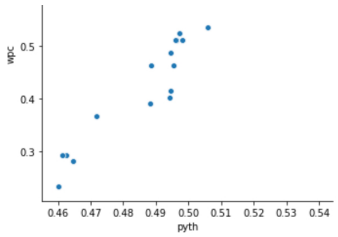
Out[8]:

	TEAM_NAME	result	PTS	PTSAGN	wpc	pyth
0	Atlanta Hawks	30	9742.0	10306.0	0.365854	0.471890
1	Boston Celtics	53	9489.0	9082.0	0.646341	0.521905
2	Brooklyn Nets	42	9375.0	9443.0	0.512195	0.496386
3	Charlotte Hornets	42	9290.0	9359.0	0.512195	0.496300
4	Chicago Bulls	24	8783.0	9467.0	0.292683	0.462573
5	Cleveland Cavaliers	24	8976.0	9697.0	0.292683	0.461446
6	Dallas Mavericks	34	8910.0	9006.0	0.414634	0.494642
7	Denver Nuggets	54	9003.0	8714.0	0.658537	0.516308
8	Detroit Pistons	43	9020.0	9069.0	0.524390	0.497291
9	Golden State Warriors	57	9677.0	9201.0	0.695122	0.525199
10	Houston Rockets	55	9356.0	8963.0	0.670732	0.521443
11	Indiana Pacers	49	9112.0	8824.0	0.597561	0.516053
12	Los Angeles Lakers	40	9506.0	9611.0	0.487805	0.494508
13	Memphis Grizzlies	38	8937.0	9145.0	0.463415	0.488498
14	Miami Heat	42	9145.0	9181.0	0.512195	0.498036
15	Milwaukee Bucks	61	9855.0	9113.0	0.743902	0.539059
16	Minnesota Timberwolves	38	9265.0	9350.0	0.463415	0.495434
17	New Orleans Pelicans	33	9589.0	9698.0	0.402439	0.494349
18	New York Knicks	19	8800.0	9531.0	0.231707	0.460186
19	Oklahoma City Thunder	50	9388.0	9091.0	0.609756	0.516068
20	Orlando Magic	44	8989.0	8885.0	0.536585	0.505818
21	Philadelphia 76ers	51	9699.0	9533.0	0.621951	0.508631
22	Phoenix Suns	23	8789.0	9432.0	0.280488	0.464755
23	Portland Trail Blazers	57	9581.0	9167.0	0.695122	0.522072
24	Sacramento Kings	42	9445.0	9521.0	0.512195	0.495993
25	San Antonio Spurs	49	9366.0	9305.0	0.597561	0.503267
26	Toronto Raptors	58	9631.0	9211.0	0.707317	0.522280
27	Utah Jazz	52	9479.0	9069.0	0.634146	0.522094
28	Washington Wizards	32	9449.0	9672.0	0.390244	0.488339

In [9]:
```
# We now plot the data. Our results look very similar to the MLB case.

sns.relplot(x="pyth", y="wpc", data = NBAteams18)
```

Out[9]:
```
<seaborn.axisgrid.FacetGrid at 0x7f5a1e44a940>
```



Self test

run sns.relplot again, but this time write y="result" instead of y="wpc". What do you find? Does it make a difference?

In [10]:
```
# Finally we run the regression: wpc = Intercept + coef x pyth
# The coefficient of the variable pyth is strongly significant, and the R-Squared of the regression is close to 100%.

pyth_lm = smf.ols(formula = 'wpc ~ pyth', data=NBAteams18).fit()
pyth_lm.summary()
```

Out[10]:

OLS Regression Results

Dep. Variable:	wpc	**R-squared:**	0.943
Model:	OLS	**Adj. R-squared:**	0.941
Method:	Least Squares	**F-statistic:**	447.0
Date:	Tue, 13 Jul 2021	**Prob (F-statistic):**	2.48e-18
Time:	04:07:05	**Log-Likelihood:**	57.497
No. Observations:	29	**AIC:**	-111.0
Df Residuals:	27	**BIC:**	-108.3
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.7572	0.155	-17.774	0.000	-3.076	-2.439

Self test

Run the regression above but instead write 'wpc ~ result' instead of 'wpc ~ result' in the line starting pyth_lm. What difference does this make?

Conclusion

We have found that the Pythagorean model fits the NBA data in roughly same way as it fits the MLB data. Let's now look at fourth example: English Premier League soccer.

In [ ]: