# Getting Started

In this series of MOOCs we aim to introduce participants to methods for analyzing sports data using Python. In this first MOOC we introduce some basic concepts. These can be broken down into three areas:

1. How to code sports data so that you can apply statistical methods
2. The use of statistical methods
3. The interpretation of results

As we go along we will introduce you to the concepts by analyzing data from different sports and generating results. Once you get the hang of how this works, you'll Pythagorean be able to do it for yourself.

In this first week, we're going to go through simple but powerful examples that introduce you to all three elements.

# The Pythagorean Expectation

The Pythagorean expectation is an idea devised by the famous baseball analyst, Bill James, but it can in fact be applied to any sport.

In any sports league, teams win games by accumulating a higher total than opponent. In baseball and cricket the relevant totals are runs, in basketball it is points, and in soccer and hockey it is goals (by "hockey" we mean here what the world outside of the US and Canada usually calls ice hockey, but in fact the same is true in field hockey).

The Pythagorean expectation can be described thus: in any season, the percentage of games won will be proportional to the square of total runs/points/goals scored by the team *squared* divided by the sum of total runs/points/goals scored by the team *squared* plus total runs/points/goals conceded by the team *squared*.

or $wpc = T_F^2 / ( T_F^2 + T_A^2 )$

Where $T_F$ is runs/points/goals scored and $T_A$ is runs/points/goals conceded.

This is a concept which can help to explain not only why teams are successful, but also can be used as the basis for predicting results in the future.

In this first week we are going to derive the Pythagorean expectation for five leagues in five different sports:

Major League Baseball The English Premier League (soccer) The Indian Premier League (cricket) The National Basketball Association (NBA) The National Hockey League (NHL)

## Coding the data

To derive the Pythagorean Expectation we will need to manipulate the data, which is a core skill that we expect you to obtain from these MOOCs. However, for this first week, we move quite quickly through the code, since our main objective is to show you the kinds of analysis you will be able to produce once you master Python.

## The Pythagorean Expectation for baseball

We begin, naturally enough, with baseball. Running code in Python typically involves the following steps:

1. Importing "packages" - these enable to run certain types of commands. The same ones come up over and over again - pandas, numpy, matplotlib.pyplot and so on.
2. Import the raw data - from a csv or excel file - in these MOOCs we will provide the data for you
3. Running commands to shape the data in preparation for running the statistical model
4. Running the statistical model
5. Reviewing the results

With each line of code below, there is a brief explanation of the code. When you are ready, read each line, then place the cursor on the relevant line and press "run" in the toolbar.

In [1]:
```
# Here are the packages we need

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]:
```
# This command imports our data, which is a log of games played in 2018 doenloaded from Retrosheet
#(you can find the data here: https://www.retrosheet.org/)
# the second line of the command prints a list of variable names - there are many more than we need

MLB = pd.read_excel('../../Data/Week 1/Retrosheet MLB game log 2018.xlsx')
print(MLB.columns.tolist())
```

```
['Date', 'DoubleHeader', 'DayOfWeek', 'VisitingTeam', 'VisitingTeamLeague', 'VisitingTeamGameNumber', 'HomeTeam', 'HomeTeamL
eague', 'HomeTeamGameNumber', 'VisitorRunsScored', 'HomeRunsScore', 'LengthInOuts', 'DayNight', 'CompletionInfo', 'ForfeitIn
fo', 'ProtestInfo', 'ParkID', 'Attendence', 'Duration', 'VisitorLineScore', 'HomeLineScore', 'VisitorAB', 'VisitorH', 'Visit
orD', 'VisitorT', 'VisitorHR', 'VisitorRBI', 'VisitorSH', 'VisitorSF', 'VisitorHBP', 'VisitorBB', 'VisitorIBB', 'VisitorK',
'VisitorSB', 'VisitorCS', 'VisitorGDP', 'VisitorCI', 'VisitorLOB', 'VisitorPitchers', 'VisitorER', 'VisitorTER', 'VisitorW
P', 'VisitorBalks', 'VisitorPO', 'VisitorA', 'VisitorE', 'VisitorPassed', 'VisitorDB', 'VisitorTP', 'HomeAB', 'HomeH', 'Home
D', 'HomeT', 'HomeHR', 'HomeRBI', 'HomeSH', 'HomeSF', 'HomeHBP', 'HomeBB', 'HomeIBB', 'HomeK', 'HomeSB', 'HomeCS', 'HomeGD
P', 'HomeCI', 'HomeLOB', 'HomePitchers', 'HomeER', 'HomeTER', 'HomeWP', 'HomeBalks', 'HomePO', 'HomeA', 'HomeE', 'HomePasse
d', 'HomeDB', 'HomeTP', 'UmpireHID', 'UmpireHName', 'Umpire1BID', 'Umpire1BName', 'Umpire2BID', 'Umpire2BName', 'Umpire3BI
D', 'Umpire3BName', 'UmpireLFID', 'UmpireLFName', 'UmpireRFID', 'UmpireRFName', 'VisitorManagerID', 'VisitorManagerName', 'H
omeManagerID', 'HomeManagerName', 'WinningPitcherID', 'WinningPitcherName', 'LosingPitcherID', 'LosingPitcherNAme', 'SavingP
itcherID', 'SavingPitcherName', 'GameWinningRBIID', 'GameWinningRBIName', 'VisitorStartingPitcherID', 'VisitorStartingPitche
rName', 'HomeStartingPitcherID', 'HomeStartingPitcherName', 'VisitorBatting1PlayerID', 'VisitorBatting1Name', 'VisitorBattin
g1Position', 'VisitorBatting2PlayerID', 'VisitorBatting2Name', 'VisitorBatting2Position', 'VisitorBatting3PlayerID', 'Visito
rBatting3Name', 'VisitorBatting3Position', 'VisitorBatting4PlayerID', 'VisitorBatting4Name', 'VisitorBatting4Position', 'Vis
itorBatting5PlayerID', 'VisitorBatting5Name', 'VisitorBatting5Position', 'VisitorBatting6PlayerID', 'VisitorBatting6Name',
'VisitorBatting6Position', 'VisitorBatting7PlayerID', 'VisitorBatting7Name', 'VisitorBatting7Position', 'VisitorBatting8Play
erID', 'VisitorBatting8Name', 'VisitorBatting8Position', 'VisitorBatting9PlayerID', 'VisitorBatting9Name', 'VisitorBatting9P
osition', 'HomeBatting1PlayerID', 'HomeBatting1Name', 'HomeBatting1Position', 'HomeBatting2PlayerID', 'HomeBatting2Name', 'H
omeBatting2Position', 'HomeBatting3PlayerID', 'HomeBatting3Name', 'HomeBatting3Position', 'HomeBatting4PlayerID', 'HomeBatti
ng4Name', 'HomeBatting4Position', 'HomeBatting5PlayerID', 'HomeBatting5Name', 'HomeBatting5Position', 'HomeBatting6PlayerI
D', 'HomeBatting6Name', 'HomeBatting6Position', 'HomeBatting7PlayerID', 'HomeBatting7Name', 'HomeBatting7Position', 'HomeBat
ting8PlayerID', 'HomeBatting8Name', 'HomeBatting8Position', 'HomeBatting9PlayerID', 'HomeBatting9Name', 'HomeBatting9Positio
n', 'AdditionalInfo', 'AcquisitionInfo']
```

In [3]:
```
# We can see what our dataframe looks like simply by typing its name
```

```
MLB
```

Out[3]:

| | Date | DoubleHeader | DayOfWeek | VisitingTeam | VisitingTeamLeague | VisitingTeamGameNumber | HomeTeam | HomeTeamLeague | HomeTeamGam |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 20180329 | 0 | Thu | COL | NL | 1 | ARI | NL | |
| 1 | 20180329 | 0 | Thu | PHI | NL | 1 | ATL | NL | |
| 2 | 20180329 | 0 | Thu | SFN | NL | 1 | LAN | NL | |
| 3 | 20180329 | 0 | Thu | CHN | NL | 1 | MIA | NL | |
| 4 | 20180329 | 0 | Thu | SLN | NL | 1 | NYN | NL | |
| 5 | 20180329 | 0 | Thu | MIL | NL | 1 | SDN | NL | |
| 6 | 20180329 | 0 | Thu | MIN | AL | 1 | BAL | AL | |
| 7 | 20180329 | 0 | Thu | CHA | AL | 1 | KCA | AL | |
| 8 | 20180329 | 0 | Thu | ANA | AL | 1 | OAK | AL | |
| 9 | 20180329 | 0 | Thu | CLE | AL | 1 | SEA | AL | |
| 10 | 20180329 | 0 | Thu | BOS | AL | 1 | TBA | AL | |

In [4]:

```
# For the Pythagorean Expectation we need only runs scored and conceded. Of course, we also need the names of the teams.
# and the date will also be useful. We put these into a new dataframe (df) which we call MLB18.
# The variable names are rather lengthy, so to make life easier we can rename columns to give them short names.
# If we want to see what the data looks like, we can just type the name of the df.

MLB18 = MLB[['VisitingTeam','HomeTeam','VisitorRunsScored','HomeRunsScore','Date']]
MLB18 = MLB18.rename(columns={'VisitorRunsScored':'VisR','HomeRunsScore':'HomR'})
MLB18
```

Out[4]:

| | VisitingTeam | HomeTeam | VisR | HomR | Date |
|---|---|---|---|---|---|
| 0 | COL | ARI | 2 | 8 | 20180329 |
| 1 | PHI | ATL | 5 | 8 | 20180329 |
| 2 | SFN | LAN | 1 | 0 | 20180329 |
| 3 | CHN | MIA | 8 | 4 | 20180329 |
| 4 | SLN | NYN | 4 | 9 | 20180329 |
| 5 | MIL | SDN | 2 | 1 | 20180329 |
| 6 | MIN | BAL | 2 | 3 | 20180329 |
| 7 | CHA | KCA | 14 | 7 | 20180329 |
| 8 | ANA | OAK | 5 | 6 | 20180329 |
| 9 | CLE | SEA | 1 | 2 | 20180329 |
| 10 | BOS | TBA | 4 | 6 | 20180329 |

In [5]:

```
# We will need to know who won the game - which we can tell by who scored the more runs, the home team or the visiting teams
#(there are no ties in baseball)
# The variable 'hwin' is defined here as equaling 1 if the home team scored more runs, and zero otherwise.
# The variable 'awin' is defined in a similar way for the away team.
# we also create a 'counter' variable = 1 for each row.

MLB18['hwin']= np.where(MLB18['HomR']>MLB18['VisR'],1,0)
MLB18['awin']= np.where(MLB18['HomR']<MLB18['VisR'],1,0)
MLB18['count']=1
MLB18
```

Out[5]:

| | VisitingTeam | HomeTeam | VisR | HomR | Date | hwin | awin | count |
|---|---|---|---|---|---|---|---|---|
| 0 | COL | ARI | 2 | 8 | 20180329 | 1 | 0 | 1 |
| 1 | PHI | ATL | 5 | 8 | 20180329 | 1 | 0 | 1 |
| 2 | SFN | LAN | 1 | 0 | 20180329 | 0 | 1 | 1 |
| 3 | CHN | MIA | 8 | 4 | 20180329 | 0 | 1 | 1 |
| 4 | SLN | NYN | 4 | 9 | 20180329 | 1 | 0 | 1 |
| 5 | MIL | SDN | 2 | 1 | 20180329 | 0 | 1 | 1 |
| 6 | MIN | BAL | 2 | 3 | 20180329 | 1 | 0 | 1 |
| 7 | CHA | KCA | 14 | 7 | 20180329 | 0 | 1 | 1 |
| 8 | ANA | OAK | 5 | 6 | 20180329 | 1 | 0 | 1 |
| 9 | CLE | SEA | 1 | 2 | 20180329 | 1 | 0 | 1 |
| 10 | BOS | TBA | 4 | 6 | 20180329 | 1 | 0 | 1 |
| 11 | HOU | TEX | 4 | 1 | 20180329 | 0 | 1 | 1 |
| 12 | NYA | TOR | 6 | 1 | 20180329 | 0 | 1 | 1 |
| 13 | COL | ARI | 8 | 9 | 20180330 | 1 | 0 | 1 |
| 14 | PHI | ATL | 5 | 4 | 20180330 | 0 | 1 | 1 |
| 15 | WAS | CIN | 2 | 0 | 20180330 | 0 | 1 | 1 |
| 16 | SFN | LAN | 1 | 0 | 20180330 | 0 | 1 | 1 |
| 17 | CHN | MIA | 1 | 2 | 20180330 | 1 | 0 | 1 |
| 18 | MIL | SDN | 8 | 6 | 20180330 | 0 | 1 | 1 |
| 19 | PIT | DET | 13 | 10 | 20180330 | 0 | 1 | 1 |
| 20 | ANA | OAK | 2 | 1 | 20180330 | 0 | 1 | 1 |
| 21 | BOS | TBA | 1 | 0 | 20180330 | 0 | 1 | 1 |
| 22 | HOU | TEX | 1 | 5 | 20180330 | 1 | 0 | 1 |
| 23 | NYA | TOR | 4 | 2 | 20180330 | 0 | 1 | 1 |
| 24 | COL | ARI | 2 | 1 | 20180331 | 0 | 1 | 1 |
| 25 | PHI | ATL | 2 | 15 | 20180331 | 1 | 0 | 1 |
| 26 | WAS | CIN | 13 | 7 | 20180331 | 0 | 1 | 1 |
| 27 | SFN | LAN | 0 | 5 | 20180331 | 1 | 0 | 1 |
| 28 | CHN | MIA | 10 | 6 | 20180331 | 0 | 1 | 1 |
| 29 | SLN | NYN | 2 | 6 | 20180331 | 1 | 0 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2401 | DET | MIL | 5 | 6 | 20180929 | 1 | 0 | 1 |
| 2402 | MIA | NYN | 0 | 1 | 20180929 | 1 | 0 | 1 |
| 2403 | ATL | PHI | 0 | 3 | 20180929 | 1 | 0 | 1 |
| 2404 | ARI | SDN | 5 | 4 | 20180929 | 0 | 1 | 1 |
| 2405 | LAN | SFN | 10 | 6 | 20180929 | 0 | 1 | 1 |
| 2406 | OAK | ANA | 5 | 2 | 20180929 | 0 | 1 | 1 |
| 2407 | HOU | BAL | 4 | 3 | 20180929 | 0 | 1 | 1 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **2408** | HOU | BAL | 5 | 2 | 20180929 | 0 | 1 | 1 |
| **2409** | NYA | BOS | 8 | 5 | 20180929 | 0 | 1 | 1 |
| **2410** | CLE | KCA | 4 | 9 | 20180929 | 1 | 0 | 1 |
| **2411** | CHA | MIN | 3 | 8 | 20180929 | 1 | 0 | 1 |
| **2412** | TEX | SEA | 1 | 4 | 20180929 | 1 | 0 | 1 |
| **2413** | TOR | TBA | 3 | 4 | 20180929 | 1 | 0 | 1 |
| **2414** | SLN | CHN | 5 | 10 | 20180930 | 1 | 0 | 1 |
| **2415** | PIT | CIN | 6 | 5 | 20180930 | 0 | 1 | 1 |
| **2416** | WAS | COL | 0 | 12 | 20180930 | 1 | 0 | 1 |
| **2417** | DET | MIL | 0 | 11 | 20180930 | 1 | 0 | 1 |
| **2418** | MIA | NYN | 0 | 1 | 20180930 | 1 | 0 | 1 |
| **2419** | ATL | PHI | 1 | 3 | 20180930 | 1 | 0 | 1 |
| **2420** | ARI | SDN | 3 | 4 | 20180930 | 1 | 0 | 1 |
| **2421** | LAN | SFN | 15 | 0 | 20180930 | 0 | 1 | 1 |
| **2422** | OAK | ANA | 4 | 5 | 20180930 | 1 | 0 | 1 |
| **2423** | HOU | BAL | 0 | 4 | 20180930 | 1 | 0 | 1 |
| **2424** | NYA | BOS | 2 | 10 | 20180930 | 1 | 0 | 1 |
| **2425** | CLE | KCA | 2 | 1 | 20180930 | 0 | 1 | 1 |
| **2426** | CHA | MIN | 4 | 5 | 20180930 | 1 | 0 | 1 |
| **2427** | TEX | SEA | 1 | 3 | 20180930 | 1 | 0 | 1 |
| **2428** | TOR | TBA | 4 | 9 | 20180930 | 1 | 0 | 1 |
| **2429** | MIL | CHN | 3 | 1 | 20181001 | 0 | 1 | 1 |
| **2430** | COL | LAN | 2 | 5 | 20181001 | 1 | 0 | 1 |

2431 rows × 8 columns

In [6]:
```
# Since our data refers to games, for each game there are two teams, but what we want is a list of runs scored and conceded
# by each team and its win percentage.
# To create this we are going to define two dfs, one for home teams and one for away teams, which we can then merge to get
# the stats for the entire season.
# Here we define a df for home teams. The command is called ".groupby" and we will use this often. We group by home team
# to obtain the sum of wins and runs (scored and conceded) and also the counter variable to show how many games were played
# (in MLB the teams do not necessarily play the same number of games in the regular season)
# Finally we rename the columns.

MLBhome = MLB18.groupby('HomeTeam')['hwin','HomR','VisR','count'].sum().reset_index()
MLBhome = MLBhome.rename(columns={'HomeTeam':'team','VisR':'VisRh','HomR':'HomRh','count':'Gh'})
MLBhome
```

Out[6]:

| | team | hwin | HomRh | VisRh | Gh |
|---|---|---|---|---|---|
| **0** | ANA | 42 | 355 | 355 | 81 |
| **1** | ARI | 40 | 359 | 328 | 81 |
| **2** | ATL | 43 | 391 | 357 | 81 |
| **3** | BAL | 28 | 339 | 411 | 81 |
| **4** | BOS | 57 | 468 | 322 | 81 |
| **5** | CHA | 30 | 321 | 409 | 81 |
| **6** | CHN | 51 | 385 | 349 | 82 |
| **7** | CIN | 37 | 385 | 418 | 81 |
| **8** | CLE | 49 | 443 | 334 | 81 |
| **9** | COL | 47 | 445 | 404 | 81 |
| **10** | DET | 38 | 330 | 363 | 81 |

In [7]:
```
#Your Code Here
```

## Self test - 1

Sometimes the code you write doesn't produce the result you want, and you need to go back and re-do it. Frequently it makes sense to go back to the beginning, rather than try to amend a df which isn't working the way you want it to. Re-starting is easy- just click on "Kernel" in the toolbar and then click "Restart and Clear Output". You can now begin again.

Copy the previous cell (first use "Insert" to add a extra cell, and then use copy and paste), and then delete ".reset_index()" and then run the code to see what happens differently. The extra headings would be a problem later on, which makes ".reset_index()" very useful in many situations.

In [8]:
```
# Now we create a similar df for teams playing as visitors - To write this code all you need to do is to copy and paste
# the previous cell and then change any reference to the home team into a reference to the visiting team.

MLBaway = MLB18.groupby('VisitingTeam')['awin','HomR','VisR','count'].sum().reset_index()
MLBaway = MLBaway.rename(columns={'VisitingTeam':'team','VisR':'VisRa','HomR':'HomRa','count':'Ga'})
MLBaway
```

Out[8]:

| | team | awin | HomRa | VisRa | Ga |
|---|---|---|---|---|---|
| **0** | ANA | 38 | 367 | 366 | 81 |
| **1** | ARI | 42 | 316 | 334 | 81 |
| **2** | ATL | 47 | 300 | 368 | 81 |
| **3** | BAL | 19 | 481 | 283 | 81 |
| **4** | BOS | 51 | 325 | 408 | 81 |
| **5** | CHA | 32 | 439 | 335 | 81 |
| **6** | CHN | 44 | 296 | 376 | 81 |
| **7** | CIN | 30 | 401 | 311 | 81 |
| **8** | CLE | 42 | 314 | 375 | 81 |
| **9** | COL | 44 | 341 | 335 | 82 |
| **10** | DET | 26 | 433 | 300 | 81 |

In [9]:
```
# We now merge MLBhome and MLBaway so that we have a list of all the clubs with home and away records for the 2018 season
# We will be using pd.merge frequently during the course to combine dfs
# Note that we've called this new df "MLB18", which is name we had already used for earlier df. By doing this we are simply
# overwriting the old MLB18 - which is fine in this case since we don't need the data in the old MLB18 any more.
# If we did want to retain the daat in the old MLB18 df, we should have given this new df a different name.

MLB18 = pd.merge(MLBhome,MLBaway,on='team')
MLB18
```

Out[9]:

| | team | hwin | HomRh | VisRh | Gh | awin | HomRa | VisRa | Ga |
|---|---|---|---|---|---|---|---|---|---|
| 0 | ANA | 42 | 355 | 355 | 81 | 38 | 367 | 366 | 81 |
| 1 | ARI | 40 | 359 | 328 | 81 | 42 | 316 | 334 | 81 |
| 2 | ATL | 43 | 391 | 357 | 81 | 47 | 300 | 368 | 81 |
| 3 | BAL | 28 | 339 | 411 | 81 | 19 | 481 | 283 | 81 |
| 4 | BOS | 57 | 468 | 322 | 81 | 51 | 325 | 408 | 81 |
| 5 | CHA | 30 | 321 | 409 | 81 | 32 | 439 | 335 | 81 |
| 6 | CHN | 51 | 385 | 349 | 82 | 44 | 296 | 376 | 81 |
| 7 | CIN | 37 | 385 | 418 | 81 | 30 | 401 | 311 | 81 |
| 8 | CLE | 49 | 443 | 334 | 81 | 42 | 314 | 375 | 81 |
| 9 | COL | 47 | 445 | 404 | 81 | 44 | 341 | 335 | 82 |
| 10 | DET | 38 | 330 | 363 | 81 | 26 | 433 | 300 | 81 |

## Self test - 2

When creating MLBhome and MLBaway we we renamed the variables using ".rename(columns ={'oldname':'newname'})". Copy and paste these cells and then re-run the code and see how the merge looks. Note that when Python encounters two variables with the same name in a merge it relabels the names with _x and _y.

Sometimes we can work with the data in this way, but usually renaming makes it easier to follow.

In [10]:
```
# Now we create the total wins, games, played, runs scored and run conceded by summing the totals as home team and away team

MLB18['W']=MLB18['hwin']+MLB18['awin']
MLB18['G']=MLB18['Gh']+MLB18['Ga']
MLB18['R']=MLB18['HomRh']+MLB18['VisRa']
MLB18['RA']=MLB18['VisRh']+MLB18['HomRa']
MLB18
```

Out[10]:

| | team | hwin | HomRh | VisRh | Gh | awin | HomRa | VisRa | Ga | W | G | R | RA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | ANA | 42 | 355 | 355 | 81 | 38 | 367 | 366 | 81 | 80 | 162 | 721 | 722 |
| 1 | ARI | 40 | 359 | 328 | 81 | 42 | 316 | 334 | 81 | 82 | 162 | 693 | 644 |
| 2 | ATL | 43 | 391 | 357 | 81 | 47 | 300 | 368 | 81 | 90 | 162 | 759 | 657 |
| 3 | BAL | 28 | 339 | 411 | 81 | 19 | 481 | 283 | 81 | 47 | 162 | 622 | 892 |
| 4 | BOS | 57 | 468 | 322 | 81 | 51 | 325 | 408 | 81 | 108 | 162 | 876 | 647 |
| 5 | CHA | 30 | 321 | 409 | 81 | 32 | 439 | 335 | 81 | 62 | 162 | 656 | 848 |
| 6 | CHN | 51 | 385 | 349 | 82 | 44 | 296 | 376 | 81 | 95 | 163 | 761 | 645 |
| 7 | CIN | 37 | 385 | 418 | 81 | 30 | 401 | 311 | 81 | 67 | 162 | 696 | 819 |
| 8 | CLE | 49 | 443 | 334 | 81 | 42 | 314 | 375 | 81 | 91 | 162 | 818 | 648 |
| 9 | COL | 47 | 445 | 404 | 81 | 44 | 341 | 335 | 82 | 91 | 163 | 780 | 745 |
| 10 | DET | 38 | 330 | 363 | 81 | 26 | 433 | 300 | 81 | 64 | 162 | 630 | 796 |

In [11]:
```
# The Last step in preparing the data is to define win percentage and the Pythagorean Expectation.

MLB18['wpc'] = MLB18['W']/MLB18['G']
MLB18['pyth'] = MLB18['R']**2/(MLB18['R']**2 + MLB18['RA']**2)
MLB18
```

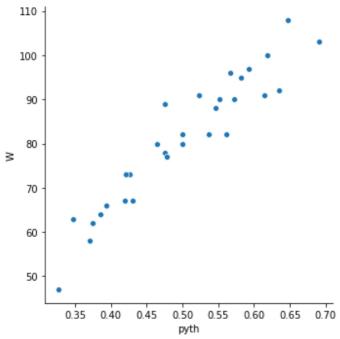
Out[11]:

| | team | hwin | HomRh | VisRh | Gh | awin | HomRa | VisRa | Ga | W | G | R | RA | wpc | pyth |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | ANA | 42 | 355 | 355 | 81 | 38 | 367 | 366 | 81 | 80 | 162 | 721 | 722 | 0.493827 | 0.499307 |
| 1 | ARI | 40 | 359 | 328 | 81 | 42 | 316 | 334 | 81 | 82 | 162 | 693 | 644 | 0.506173 | 0.536600 |
| 2 | ATL | 43 | 391 | 357 | 81 | 47 | 300 | 368 | 81 | 90 | 162 | 759 | 657 | 0.555556 | 0.571662 |
| 3 | BAL | 28 | 339 | 411 | 81 | 19 | 481 | 283 | 81 | 47 | 162 | 622 | 892 | 0.290123 | 0.327161 |
| 4 | BOS | 57 | 468 | 322 | 81 | 51 | 325 | 408 | 81 | 108 | 162 | 876 | 647 | 0.666667 | 0.647037 |
| 5 | CHA | 30 | 321 | 409 | 81 | 32 | 439 | 335 | 81 | 62 | 162 | 656 | 848 | 0.382716 | 0.374388 |
| 6 | CHN | 51 | 385 | 349 | 82 | 44 | 296 | 376 | 81 | 95 | 163 | 761 | 645 | 0.582822 | 0.581946 |
| 7 | CIN | 37 | 385 | 418 | 81 | 30 | 401 | 311 | 81 | 67 | 162 | 696 | 819 | 0.413580 | 0.419344 |
| 8 | CLE | 49 | 443 | 334 | 81 | 42 | 314 | 375 | 81 | 91 | 162 | 818 | 648 | 0.561728 | 0.614423 |
| 9 | COL | 47 | 445 | 404 | 81 | 44 | 341 | 335 | 82 | 91 | 163 | 780 | 745 | 0.558282 | 0.522939 |
| 10 | DET | 38 | 330 | 363 | 81 | 26 | 433 | 300 | 81 | 64 | 162 | 630 | 796 | 0.395062 | 0.385147 |
| 11 | HOU | 46 | 373 | 288 | 81 | 57 | 246 | 424 | 81 | 103 | 162 | 797 | 534 | 0.635802 | 0.690171 |
| 12 | KCA | 32 | 333 | 424 | 81 | 26 | 409 | 305 | 81 | 58 | 162 | 638 | 833 | 0.358025 | 0.369726 |
| 13 | LAN | 45 | 366 | 297 | 82 | 47 | 313 | 438 | 81 | 92 | 163 | 804 | 610 | 0.564417 | 0.634665 |
| 14 | MIA | 38 | 279 | 323 | 81 | 25 | 486 | 310 | 80 | 63 | 161 | 589 | 809 | 0.391304 | 0.346435 |
| 15 | MIL | 51 | 384 | 322 | 81 | 45 | 337 | 370 | 82 | 96 | 163 | 754 | 659 | 0.588957 | 0.566930 |
| 16 | MIN | 49 | 397 | 361 | 81 | 29 | 414 | 341 | 81 | 78 | 162 | 738 | 775 | 0.481481 | 0.475560 |
| 17 | NYA | 53 | 453 | 352 | 81 | 47 | 317 | 398 | 81 | 100 | 162 | 851 | 669 | 0.617284 | 0.618044 |
| 18 | NYN | 37 | 274 | 310 | 81 | 40 | 397 | 402 | 81 | 77 | 162 | 676 | 707 | 0.475309 | 0.477596 |
| 19 | OAK | 50 | 369 | 310 | 81 | 47 | 364 | 444 | 81 | 97 | 162 | 813 | 674 | 0.598765 | 0.592667 |
| 20 | PHI | 49 | 370 | 347 | 81 | 31 | 381 | 307 | 81 | 80 | 162 | 677 | 728 | 0.493827 | 0.463749 |
| 21 | PIT | 44 | 326 | 318 | 80 | 38 | 375 | 366 | 81 | 82 | 161 | 692 | 693 | 0.509317 | 0.499278 |
| 22 | SDN | 31 | 313 | 390 | 81 | 35 | 377 | 304 | 81 | 66 | 162 | 617 | 767 | 0.407407 | 0.392877 |
| 23 | SEA | 45 | 299 | 337 | 81 | 44 | 374 | 378 | 81 | 89 | 162 | 677 | 711 | 0.549383 | 0.475519 |
| 24 | SFN | 42 | 321 | 334 | 81 | 31 | 365 | 282 | 81 | 73 | 162 | 603 | 699 | 0.450617 | 0.426666 |
| 25 | SLN | 43 | 351 | 346 | 81 | 45 | 345 | 408 | 81 | 88 | 162 | 759 | 691 | 0.543210 | 0.546794 |
| 26 | TBA | 51 | 371 | 284 | 81 | 39 | 362 | 345 | 81 | 90 | 162 | 716 | 646 | 0.555556 | 0.551260 |
| 27 | TEX | 34 | 432 | 479 | 81 | 33 | 369 | 305 | 81 | 67 | 162 | 737 | 848 | 0.413580 | 0.430310 |
| 28 | TOR | 40 | 361 | 393 | 81 | 33 | 439 | 348 | 81 | 73 | 162 | 709 | 832 | 0.450617 | 0.420687 |
| 29 | WAS | 41 | 409 | 363 | 81 | 41 | 319 | 362 | 81 | 82 | 162 | 771 | 682 | 0.506173 | 0.561024 |

In [12]:
```
# Having prepared the data, we are now ready to examine it. First, we generate and xy plot use the Seaborn package.
# This illustrates nicely the close correlation between win percentage and the Pythagorean Expectation.

sns.relplot(x="pyth", y="W", data = MLB18)
```

Out[12]:
```
<seaborn.axisgrid.FacetGrid at 0x7f942b2a6940>
```




### Self test - 3

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

## Finally we generate a regression.

The regression output tells you many things about the fitted relationship between win percentage and the Pythagorean Expectation. Regression is a method for identifying an equation which best fits the data. In this case that relationship is

wpc = Intercept + coef x pyth

You can see the value of Intercept is 0.0609 and coef is .8770. It's this latter value were interested in. It means that for every one unit increase in pyth, the value of wpc goes up by 0.887.

Two other points to note:

(i) The standard error (std err) gives us an idea of the precision of the estimate. The ratio of the coefficient (coef) to the standard error is called the t statistic (t) and its value informs us about statistical significance. This is illustrated by the p-value (P > |t|) - this is the probability that we would observe the value .8770 by chance, if the true value were really zero. This probability here is 0.000 - (this is not exactly zero, but the table doesn't include enough decimal places to show this) which means we can confident it is not zero. By convention, it is usual to conclude that we cannot be confident that the value of the coefficient is not zero if the p-value is greater than .05

(ii) in the top right hand corner of the table is the R-squared. This statistic tells you the percentage of variation in the y-variable (wpc) which can be accounted for by the variation in the x variables (pyth). R-squared can be thought of as a percentage - here the Pythagorean Expectation can account for 89.4% of the variation in win percentage.

In [13]:
```
# Finally we generate a regression.

pyth_lm = smf.ols(formula = 'wpc ~ pyth', data=MLB18).fit()
pyth_lm.summary()
```

Out[13]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | wpc | **R-squared:** | 0.894 |
| **Model:** | OLS | **Adj. R-squared:** | 0.890 |
| **Method:** | Least Squares | **F-statistic:** | 236.2 |
| **Date:** | Tue, 13 Jul 2021 | **Prob (F-statistic):** | 3.54e-15 |
| **Time:** | 03:43:22 | **Log-Likelihood:** | 63.733 |
| **No. Observations:** | 30 | **AIC:** | -123.5 |
| **Df Residuals:** | 28 | **BIC:** | -120.7 |
| **Df Model:** | 1 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | 0.0609 | 0.029 | 2.093 | 0.046 | 0.001 | 0.120 |
| **pyth** | 0.8770 | 0.057 | 15.370 | 0.000 | 0.760 | 0.994 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 0.145 | **Durbin-Watson:** | 1.987 |
| **Prob(Omnibus):** | 0.930 | **Jarque-Bera (JB):** | 0.012 |
| **Skew:** | -0.009 | **Prob(JB):** | 0.994 |
| **Kurtosis:** | 2.905 | **Cond. No.** | 13.1 |

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

### Self test - 4

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

## Conclusion

This example was intended to get you started- don't worry if some things seem unclear - we're now going to conduct the same analysis for cricket, basketball, soccer and hockey. This will extend your understanding and help to make clear what we have just looked at.

A Useful Tip: when working in Python you will often come across problems that can be solved using methods you have encountered previously. It is often a good idea to return to an old notebook at a later stage to remind yourself how to code a particular problem.

In [ ]: