1 Pythagorean expectation and MLB

17 July 2020

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 = TF2 / (TF2 + TA2)*

Where TF is runs/points/goals scored and TA 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.

```
# Here are the packages we need
library("readxl",quietly = TRUE)
library("tidyverse",quietly = TRUE)
```

```
# 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  <- read_excel("Retrosheet MLB game log 2018.xlsx")
```

```
## Warning in read_fun(path = enc2native(normalizePath(path)), sheet_i =
## sheet, : Expecting logical in CH2431 / R2431C86: got 'reybd901'
```

```
## Warning in read_fun(path = enc2native(normalizePath(path)), sheet_i =
## sheet, : Expecting logical in CJ2431 / R2431C88: got 'hamaa901'
```

```
## Warning in read_fun(path = enc2native(normalizePath(path)), sheet_i =
## sheet, : Expecting logical in CH2432 / R2432C86: got 'rackd901'
```

```
## Warning in read_fun(path = enc2native(normalizePath(path)), sheet_i =
## sheet, : Expecting logical in CJ2432 / R2432C88: got 'wolcq901'
```

```
names(MLB)
```

```
##   [1] "Date"                    "DoubleHeader"
##   [3] "DayOfWeek"               "VisitingTeam"
##   [5] "VisitingTeamLeague"      "VisitingTeamGameNumber"
##   [7] "HomeTeam"                "HomeTeamLeague"
##   [9] "HomeTeamGameNumber"      "VisitorRunsScored"
##  [11] "HomeRunsScore"           "LengthInOuts"
##  [13] "DayNight"                "CompletionInfo"
##  [15] "ForfeitInfo"             "ProtestInfo"
##  [17] "ParkID"                  "Attendence"
##  [19] "Duration"                "VisitorLineScore"
##  [21] "HomeLineScore"           "VisitorAB"
##  [23] "VisitorH"                "VisitorD"
##  [25] "VisitorT"                "VisitorHR"
##  [27] "VisitorRBI"              "VisitorSH"
##  [29] "VisitorSF"               "VisitorHBP"
##  [31] "VisitorBB"               "VisitorIBB"
##  [33] "VisitorK"                "VisitorSB"
##  [35] "VisitorCS"               "VisitorGDP"
##  [37] "VisitorCI"               "VisitorLOB"
##  [39] "VisitorPitchers"         "VisitorER"
##  [41] "VisitorTER"              "VisitorWP"
##  [43] "VisitorBalks"            "VisitorPO"
##  [45] "VisitorA"                "VisitorE"
##  [47] "VisitorPassed"           "VisitorDB"
##  [49] "VisitorTP"               "HomeAB"
##  [51] "HomeH"                   "HomeD"
##  [53] "HomeT"                   "HomeHR"
##  [55] "HomeRBI"                 "HomeSH"
##  [57] "HomeSF"                  "HomeHBP"
##  [59] "HomeBB"                  "HomeIBB"
##  [61] "HomeK"                   "HomeSB"
##  [63] "HomeCS"                  "HomeGDP"
##  [65] "HomeCI"                  "HomeLOB"
##  [67] "HomePitchers"            "HomeER"
##  [69] "HomeTER"                 "HomeWP"
##  [71] "HomeBalks"               "HomePO"
##  [73] "HomeA"                   "HomeE"
##  [75] "HomePassed"              "HomeDB"
##  [77] "HomeTP"                  "UmpireHID"
##  [79] "UmpireHName"             "Umpire1BID"
```

```
##  [81] "Umpire1BName"               "Umpire2BID"
##  [83] "Umpire2BName"               "Umpire3BID"
##  [85] "Umpire3BName"               "UmpireLFID"
##  [87] "UmpireLFName"               "UmpireRFID"
##  [89] "UmpireRFName"               "VisitorManagerID"
##  [91] "VisitorManagerName"         "HomeManagerID"
##  [93] "HomeManagerName"            "WinningPitcherID"
##  [95] "WinningPitcherName"         "LosingPitcherID"
##  [97] "LosingPitcherNAme"          "SavingPitcherID"
##  [99] "SavingPitcherName"          "GameWinningRBIID"
## [101] "GameWinningRBIName"         "VisitorStartingPitcherID"
## [103] "VisitorStartingPitcherName" "HomeStartingPitcherID"
## [105] "HomeStartingPitcherName"    "VisitorBatting1PlayerID"
## [107] "VisitorBatting1Name"        "VisitorBatting1Position"
## [109] "VisitorBatting2PlayerID"    "VisitorBatting2Name"
## [111] "VisitorBatting2Position"    "VisitorBatting3PlayerID"
## [113] "VisitorBatting3Name"        "VisitorBatting3Position"
## [115] "VisitorBatting4PlayerID"    "VisitorBatting4Name"
## [117] "VisitorBatting4Position"    "VisitorBatting5PlayerID"
## [119] "VisitorBatting5Name"        "VisitorBatting5Position"
## [121] "VisitorBatting6PlayerID"    "VisitorBatting6Name"
## [123] "VisitorBatting6Position"    "VisitorBatting7PlayerID"
## [125] "VisitorBatting7Name"        "VisitorBatting7Position"
## [127] "VisitorBatting8PlayerID"    "VisitorBatting8Name"
## [129] "VisitorBatting8Position"    "VisitorBatting9PlayerID"
## [131] "VisitorBatting9Name"        "VisitorBatting9Position"
## [133] "HomeBatting1PlayerID"       "HomeBatting1Name"
## [135] "HomeBatting1Position"       "HomeBatting2PlayerID"
## [137] "HomeBatting2Name"           "HomeBatting2Position"
## [139] "HomeBatting3PlayerID"       "HomeBatting3Name"
## [141] "HomeBatting3Position"       "HomeBatting4PlayerID"
## [143] "HomeBatting4Name"           "HomeBatting4Position"
## [145] "HomeBatting5PlayerID"       "HomeBatting5Name"
## [147] "HomeBatting5Position"       "HomeBatting6PlayerID"
## [149] "HomeBatting6Name"           "HomeBatting6Position"
## [151] "HomeBatting7PlayerID"       "HomeBatting7Name"
## [153] "HomeBatting7Position"       "HomeBatting8PlayerID"
## [155] "HomeBatting8Name"           "HomeBatting8Position"
## [157] "HomeBatting9PlayerID"       "HomeBatting9Name"
## [159] "HomeBatting9Position"       "AdditionalInfo"
## [161] "AcquisitionInfo"
```

```
# We can see what our dataframe looks like simply by typing its name

MLB
```

```
## # A tibble: 2,431 x 161
##      Date DoubleHeader DayOfWeek VisitingTeam VisitingTeamLea~
##     <dbl>        <dbl> <chr>     <chr>        <chr>
##  1 2.02e7            0 Thu       COL          NL
##  2 2.02e7            0 Thu       PHI          NL
##  3 2.02e7            0 Thu       SFN          NL
##  4 2.02e7            0 Thu       CHN          NL
##  5 2.02e7            0 Thu       SLN          NL
##  6 2.02e7            0 Thu       MIL          NL
##  7 2.02e7            0 Thu       MIN          AL
##  8 2.02e7            0 Thu       CHA          AL
##  9 2.02e7            0 Thu       ANA          AL
## 10 2.02e7            0 Thu       CLE          AL
## # ... with 2,421 more rows, and 156 more variables:
## #   VisitingTeamGameNumber <dbl>, HomeTeam <chr>, HomeTeamLeague <chr>,
## #   HomeTeamGameNumber <dbl>, VisitorRunsScored <dbl>,
## #   HomeRunsScore <dbl>, LengthInOuts <dbl>, DayNight <chr>,
## #   CompletionInfo <chr>, ForfeitInfo <lgl>, ProtestInfo <lgl>,
## #   ParkID <chr>, Attendence <dbl>, Duration <dbl>,
## #   VisitorLineScore <chr>, HomeLineScore <chr>, VisitorAB <dbl>,
## #   VisitorH <dbl>, VisitorD <dbl>, VisitorT <dbl>, VisitorHR <dbl>,
## #   VisitorRBI <dbl>, VisitorSH <dbl>, VisitorSF <dbl>, VisitorHBP <dbl>,
## #   VisitorBB <dbl>, VisitorIBB <dbl>, VisitorK <dbl>, VisitorSB <dbl>,
## #   VisitorCS <dbl>, VisitorGDP <dbl>, VisitorCI <dbl>, VisitorLOB <dbl>,
## #   VisitorPitchers <dbl>, VisitorER <dbl>, VisitorTER <dbl>,
## #   VisitorWP <dbl>, VisitorBalks <dbl>, VisitorPO <dbl>, VisitorA <dbl>,
## #   VisitorE <dbl>, VisitorPassed <dbl>, VisitorDB <dbl>, VisitorTP <dbl>,
## #   HomeAB <dbl>, HomeH <dbl>, HomeD <dbl>, HomeT <dbl>, HomeHR <dbl>,
## #   HomeRBI <dbl>, HomeSH <dbl>, HomeSF <dbl>, HomeHBP <dbl>,
## #   HomeBB <dbl>, HomeIBB <dbl>, HomeK <dbl>, HomeSB <dbl>, HomeCS <dbl>,
## #   HomeGDP <dbl>, HomeCI <dbl>, HomeLOB <dbl>, HomePitchers <dbl>,
## #   HomeER <dbl>, HomeTER <dbl>, HomeWP <dbl>, HomeBalks <dbl>,
## #   HomePO <dbl>, HomeA <dbl>, HomeE <dbl>, HomePassed <dbl>,
## #   HomeDB <dbl>, HomeTP <dbl>, UmpireHID <chr>, UmpireHName <chr>,
## #   Umpire1BID <chr>, Umpire1BName <chr>, Umpire2BID <chr>,
## #   Umpire2BName <chr>, Umpire3BID <chr>, Umpire3BName <chr>,
## #   UmpireLFID <lgl>, UmpireLFName <chr>, UmpireRFID <lgl>,
## #   UmpireRFName <chr>, VisitorManagerID <chr>, VisitorManagerName <chr>,
## #   HomeManagerID <chr>, HomeManagerName <chr>, WinningPitcherID <chr>,
## #   WinningPitcherName <chr>, LosingPitcherID <chr>,
## #   LosingPitcherNAme <chr>, SavingPitcherID <chr>,
## #   SavingPitcherName <chr>, GameWinningRBIID <chr>,
## #   GameWinningRBIName <chr>, VisitorStartingPitcherID <chr>,
## #   VisitorStartingPitcherName <chr>, HomeStartingPitcherID <chr>,
## #   HomeStartingPitcherName <chr>, ...
```

```
# 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[,c('VisitingTeam','HomeTeam','VisitorRunsScored','HomeRunsScore','Date')]
MLB18 <- MLB18 %>%
  rename(VisR = VisitorRunsScored, HomR = HomeRunsScore)
MLB18
```

```
## # A tibble: 2,431 x 5
##    VisitingTeam HomeTeam  VisR  HomR     Date
##    <chr>        <chr>    <dbl> <dbl>    <dbl>
##  1 COL          ARI          2     8 20180329
##  2 PHI          ATL          5     8 20180329
##  3 SFN          LAN          1     0 20180329
##  4 CHN          MIA          8     4 20180329
##  5 SLN          NYN          4     9 20180329
##  6 MIL          SDN          2     1 20180329
##  7 MIN          BAL          2     3 20180329
##  8 CHA          KCA         14     7 20180329
##  9 ANA          OAK          5     6 20180329
## 10 CLE          SEA          1     2 20180329
## # ... with 2,421 more rows
```

```
# 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$hwin = ifelse(MLB18$HomR > MLB18$VisR,1,0)
MLB18$awin = ifelse(MLB18$HomR < MLB18$VisR,1,0)
MLB18$count = 1
MLB18
```

```
## # A tibble: 2,431 x 8
##    VisitingTeam HomeTeam  VisR  HomR     Date  hwin  awin count
##    <chr>        <chr>    <dbl> <dbl>    <dbl> <dbl> <dbl> <dbl>
##  1 COL          ARI          2     8 20180329     1     0     1
##  2 PHI          ATL          5     8 20180329     1     0     1
```

```
##  3 SFN          LAN           1     0 20180329     0     1     1
##  4 CHN          MIA           8     4 20180329     0     1     1
##  5 SLN          NYN           4     9 20180329     1     0     1
##  6 MIL          SDN           2     1 20180329     0     1     1
##  7 MIN          BAL           2     3 20180329     1     0     1
##  8 CHA          KCA          14     7 20180329     0     1     1
##  9 ANA          OAK           5     6 20180329     1     0     1
## 10 CLE          SEA           1     2 20180329     1     0     1
## # ... with 2,421 more rows
```

```
# 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 "group_by()"
# 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 %>% group_by(HomeTeam)%>%
            dplyr::summarise(hwin= sum(hwin),
                             HomR = sum(HomR),
                             VisR = sum(VisR),
                             count = sum(count),
                            )%>%
                            ungroup()%>%
                            rename(team = HomeTeam,
                                   VisRh = VisR,
                                   HomRh = HomR,
                                   Gh = count)
MLBhome
```

```
## # A tibble: 30 x 5
##    team   hwin HomRh VisRh    Gh
##    <chr> <dbl> <dbl> <dbl> <dbl>
##  1 ANA      42   355   355    81
##  2 ARI      40   359   328    81
##  3 ATL      43   391   357    81
##  4 BAL      28   339   411    81
##  5 BOS      57   468   322    81
```

```
##  6 CHA      30   321   409    81
##  7 CHN      51   385   349    82
##  8 CIN      37   385   418    81
##  9 CLE      49   443   334    81
## 10 COL      47   445   404    81
## # ... with 20 more rows
```

Self test

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.

```
# 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.
str(MLB18)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    2431 obs. of  8 variables:
##  $ VisitingTeam: chr  "COL" "PHI" "SFN" "CHN" ...
##  $ HomeTeam    : chr  "ARI" "ATL" "LAN" "MIA" ...
##  $ VisR        : num  2 5 1 8 4 2 2 14 5 1 ...
##  $ HomR        : num  8 8 0 4 9 1 3 7 6 2 ...
##  $ Date        : num  20180329 20180329 20180329 20180329 20180329 ...
##  $ hwin        : num  1 1 0 0 1 0 1 0 1 1 ...
##  $ awin        : num  0 0 1 1 0 1 0 1 0 0 ...
##  $ count       : num  1 1 1 1 1 1 1 1 1 1 ...
```

```
MLBaway <- MLB18 %>% group_by(VisitingTeam)%>%
            dplyr::summarise(awin= sum(awin),
                             HomR = sum(HomR),
                             VisR = sum(VisR),
                             count = sum(count),
                            )%>%
                            ungroup()%>%
                            rename(team = VisitingTeam,
                                   VisRa = VisR,
                                   HomRa = HomR,
                                   Ga = count)
```

```
MLBaway
```

```
## # A tibble: 30 x 5
##    team   awin HomRa VisRa    Ga
##    <chr> <dbl> <dbl> <dbl> <dbl>
##  1 ANA      38   367   366    81
##  2 ARI      42   316   334    81
##  3 ATL      47   300   368    81
##  4 BAL      19   481   283    81
##  5 BOS      51   325   408    81
##  6 CHA      32   439   335    81
##  7 CHN      44   296   376    81
##  8 CIN      30   401   311    81
##  9 CLE      42   314   375    81
## 10 COL      44   341   335    82
## # ... with 20 more rows
```

```
# 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 <- merge(x=MLBhome,y=MLBaway,by=c('team'))
MLB18
```

```
##    team hwin HomRh VisRh Gh awin HomRa VisRa Ga
## 1   ANA   42   355   355 81   38   367   366 81
## 2   ARI   40   359   328 81   42   316   334 81
## 3   ATL   43   391   357 81   47   300   368 81
## 4   BAL   28   339   411 81   19   481   283 81
## 5   BOS   57   468   322 81   51   325   408 81
## 6   CHA   30   321   409 81   32   439   335 81
## 7   CHN   51   385   349 82   44   296   376 81
## 8   CIN   37   385   418 81   30   401   311 81
## 9   CLE   49   443   334 81   42   314   375 81
## 10  COL   47   445   404 81   44   341   335 82
## 11  DET   38   330   363 81   26   433   300 81
## 12  HOU   46   373   288 81   57   246   424 81
## 13  KCA   32   333   424 81   26   409   305 81
```

```
## 14  LAN   45   366   297 82   47   313   438 81
## 15  MIA   38   279   323 81   25   486   310 80
## 16  MIL   51   384   322 81   45   337   370 82
## 17  MIN   49   397   361 81   29   414   341 81
## 18  NYA   53   453   352 81   47   317   398 81
## 19  NYN   37   274   310 81   40   397   402 81
## 20  OAK   50   369   310 81   47   364   444 81
## 21  PHI   49   370   347 81   31   381   307 81
## 22  PIT   44   326   318 80   38   375   366 81
## 23  SDN   31   313   390 81   35   377   304 81
## 24  SEA   45   299   337 81   44   374   378 81
## 25  SFN   42   321   334 81   31   365   282 81
## 26  SLN   43   351   346 81   45   345   408 81
## 27  TBA   51   371   284 81   39   362   345 81
## 28  TEX   34   432   479 81   33   369   305 81
## 29  TOR   40   361   393 81   33   439   348 81
## 30  WAS   41   409   363 81   41   319   362 81
```

Self test

When creating MLBhome and MLBaway we we renamed the variables using "rename(newname = oldname). Copy and paste these cells and then re-run the code and see how the merge looks. Note that when R 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.

```
# 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
```

```
##    team hwin HomRh VisRh Gh awin HomRa VisRa Ga   W   G   R  RA
## 1   ANA   42   355   355 81   38   367   366 81  80 162 721 722
## 2   ARI   40   359   328 81   42   316   334 81  82 162 693 644
## 3   ATL   43   391   357 81   47   300   368 81  90 162 759 657
## 4   BAL   28   339   411 81   19   481   283 81  47 162 622 892
## 5   BOS   57   468   322 81   51   325   408 81 108 162 876 647
## 6   CHA   30   321   409 81   32   439   335 81  62 162 656 848
## 7   CHN   51   385   349 82   44   296   376 81  95 163 761 645
## 8   CIN   37   385   418 81   30   401   311 81  67 162 696 819
```

```
## 9   CLE   49   443   334 81   42   314   375 81  91 162 818 648
## 10  COL   47   445   404 81   44   341   335 82  91 163 780 745
## 11  DET   38   330   363 81   26   433   300 81  64 162 630 796
## 12  HOU   46   373   288 81   57   246   424 81 103 162 797 534
## 13  KCA   32   333   424 81   26   409   305 81  58 162 638 833
## 14  LAN   45   366   297 82   47   313   438 81  92 163 804 610
## 15  MIA   38   279   323 81   25   486   310 80  63 161 589 809
## 16  MIL   51   384   322 81   45   337   370 82  96 163 754 659
## 17  MIN   49   397   361 81   29   414   341 81  78 162 738 775
## 18  NYA   53   453   352 81   47   317   398 81 100 162 851 669
## 19  NYN   37   274   310 81   40   397   402 81  77 162 676 707
## 20  OAK   50   369   310 81   47   364   444 81  97 162 813 674
## 21  PHI   49   370   347 81   31   381   307 81  80 162 677 728
## 22  PIT   44   326   318 80   38   375   366 81  82 161 692 693
## 23  SDN   31   313   390 81   35   377   304 81  66 162 617 767
## 24  SEA   45   299   337 81   44   374   378 81  89 162 677 711
## 25  SFN   42   321   334 81   31   365   282 81  73 162 603 699
## 26  SLN   43   351   346 81   45   345   408 81  88 162 759 691
## 27  TBA   51   371   284 81   39   362   345 81  90 162 716 646
## 28  TEX   34   432   479 81   33   369   305 81  67 162 737 848
## 29  TOR   40   361   393 81   33   439   348 81  73 162 709 832
## 30  WAS   41   409   363 81   41   319   362 81  82 162 771 682
```

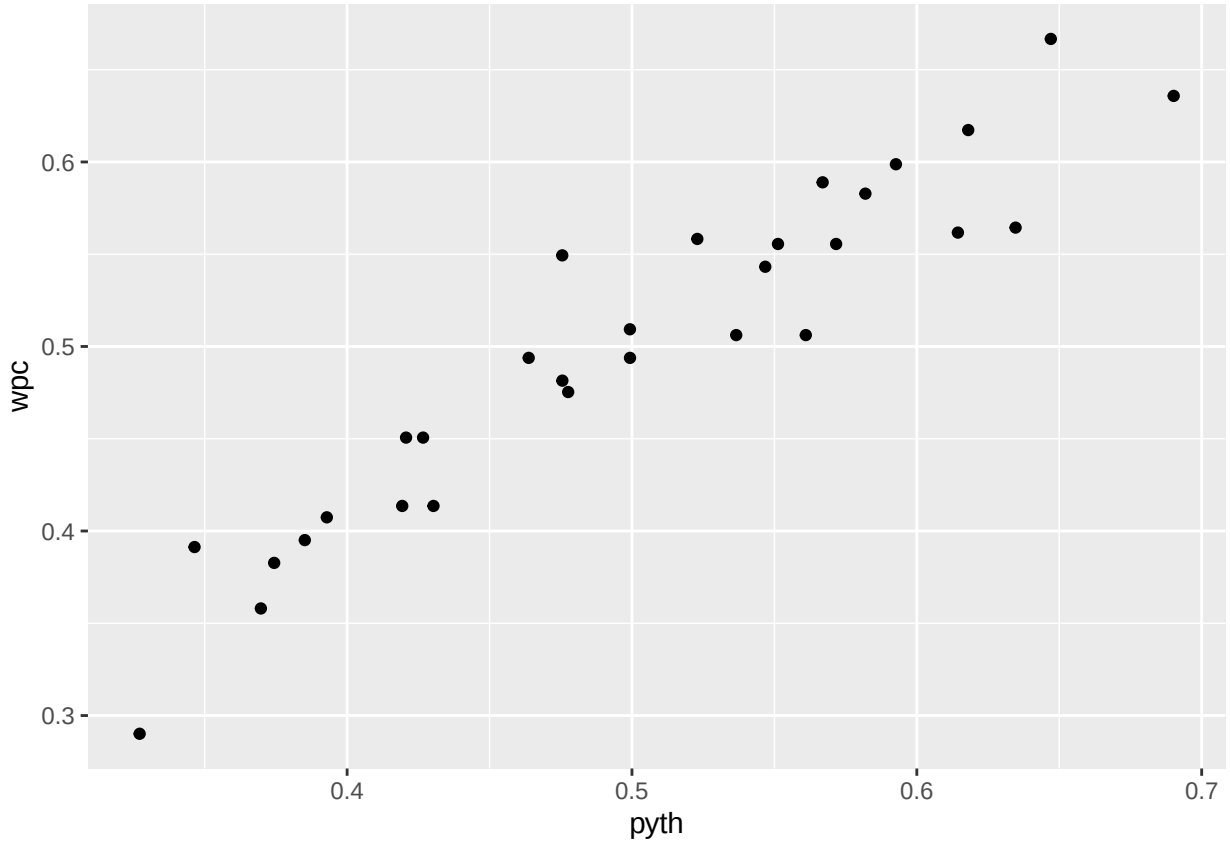
```
# The last step in preparing the data is to define win percentage
# and the Pythagorean Expectation.

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

```
##    team hwin HomRh VisRh Gh awin HomRa VisRa Ga   W   G   R  RA       wpc
## 1   ANA   42   355   355 81   38   367   366 81  80 162 721 722 0.4938272
## 2   ARI   40   359   328 81   42   316   334 81  82 162 693 644 0.5061728
## 3   ATL   43   391   357 81   47   300   368 81  90 162 759 657 0.5555556
## 4   BAL   28   339   411 81   19   481   283 81  47 162 622 892 0.2901235
## 5   BOS   57   468   322 81   51   325   408 81 108 162 876 647 0.6666667
## 6   CHA   30   321   409 81   32   439   335 81  62 162 656 848 0.3827160
## 7   CHN   51   385   349 82   44   296   376 81  95 163 761 645 0.5828221
## 8   CIN   37   385   418 81   30   401   311 81  67 162 696 819 0.4135802
## 9   CLE   49   443   334 81   42   314   375 81  91 162 818 648 0.5617284
## 10  COL   47   445   404 81   44   341   335 82  91 163 780 745 0.5582822
## 11  DET   38   330   363 81   26   433   300 81  64 162 630 796 0.3950617
## 12  HOU   46   373   288 81   57   246   424 81 103 162 797 534 0.6358025
## 13  KCA   32   333   424 81   26   409   305 81  58 162 638 833 0.3580247
## 14  LAN   45   366   297 82   47   313   438 81  92 163 804 610 0.5644172
## 15  MIA   38   279   323 81   25   486   310 80  63 161 589 809 0.3913043
```

```
## 16  MIL   51   384   322 81   45   337   370 82  96 163 754 659 0.5889571
## 17  MIN   49   397   361 81   29   414   341 81  78 162 738 775 0.4814815
## 18  NYA   53   453   352 81   47   317   398 81 100 162 851 669 0.6172840
## 19  NYN   37   274   310 81   40   397   402 81  77 162 676 707 0.4753086
## 20  OAK   50   369   310 81   47   364   444 81  97 162 813 674 0.5987654
## 21  PHI   49   370   347 81   31   381   307 81  80 162 677 728 0.4938272
## 22  PIT   44   326   318 80   38   375   366 81  82 161 692 693 0.5093168
## 23  SDN   31   313   390 81   35   377   304 81  66 162 617 767 0.4074074
## 24  SEA   45   299   337 81   44   374   378 81  89 162 677 711 0.5493827
## 25  SFN   42   321   334 81   31   365   282 81  73 162 603 699 0.4506173
## 26  SLN   43   351   346 81   45   345   408 81  88 162 759 691 0.5432099
## 27  TBA   51   371   284 81   39   362   345 81  90 162 716 646 0.5555556
## 28  TEX   34   432   479 81   33   369   305 81  67 162 737 848 0.4135802
## 29  TOR   40   361   393 81   33   439   348 81  73 162 709 832 0.4506173
## 30  WAS   41   409   363 81   41   319   362 81  82 162 771 682 0.5061728
##         pyth
## 1  0.4993070
## 2  0.5366001
## 3  0.5716621
## 4  0.3271613
## 5  0.6470369
## 6  0.3743875
## 7  0.5819458
## 8  0.4193435
## 9  0.6144231
## 10 0.5229387
## 11 0.3851469
## 12 0.6901707
## 13 0.3697264
## 14 0.6346646
## 15 0.3464353
## 16 0.5669303
## 17 0.4755599
## 18 0.6180444
## 19 0.4775962
## 20 0.5926671
## 21 0.4637488
## 22 0.4992780
## 23 0.3928768
## 24 0.4755190
## 25 0.4266660
## 26 0.5467936
## 27 0.5512596
## 28 0.4303102
## 29 0.4206870
```

```
## 30 0.5610236
```

```
# 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.
ggplot(data = MLB18,aes(x = pyth,y = wpc )) + geom_point()
```



Self test

run ggplot 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.

```
# Finally we generate a regression.

pyth_lm = lm(formula = 'wpc ~ pyth', data = MLB18)
pyth_lm %>% summary()
```

```
##
## Call:
## lm(formula = "wpc ~ pyth", data = MLB18)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.057642 -0.022238  0.002425  0.017520  0.071514
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.06086    0.02908   2.093   0.0456 *
## pyth         0.87695    0.05706  15.370 3.54e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02993 on 28 degrees of freedom
## Multiple R-squared:  0.894,  Adjusted R-squared:  0.8902
## F-statistic: 236.2 on 1 and 28 DF,  p-value: 3.544e-15
```

Self test

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.