Introduction:

The thing of beauty in baseball is that each year we have a chance to see players making a leap. Like Jose Bautista, Jose Ramirez, Ben Zobrist, etc. This research aims to find out of these breakout players, the improvement of what stats are more responsible for their WAR and wRC+ gain.

Methods:

Data include MLB players stats from 2000-2020
During that period, 63 players who had a leap year were selected
Not only look at players who had a leap(defined as having a 4 war differential from last season), but also verify that leap by looking at the two years average before that leap year
For example: 2007 Ben Zobrist had a -1.1 WAR season, jumped to 1.5 in 2011 then 8.7 in 2012. He had a 7.2(8.7-1.5) WAR jump as well as verified by 8.7-(-1.1+1.5)/2=8.5(marked as jumpfromavg in column below)
Batting categories include G(games played), PA, HR, R, SB, BBp(BB percentage), Kp(K percentage), ISO, BABIP, AVG, OBP, SLG, wOBA, wRCp(wRC+), BsR(Base Running), GBFB(GB/FB), Batted ball type(LDp, GBp, FBp), IFFBp(Infield Fly Ball percentage), HRFB(HR/FB), IFHp(Infield hit percentage), Batted ball direction(Pullp, Centp, Oppop), Quality of contact(Softp, Medp, Hardp).

Step1: Import the data of the year before jump

In [1]:

```
import pandas as pd
import pyodbc

#connect with sql server and retrieve the data we want
sql_conn = pyodbc.connect('''DRIVER={ODBC Driver 13 for SQL Server};
                            SERVER=ALLENHO\MSSQLSERVER002;
                            DATABASE=WAR Jump;
                            Trusted_Connection=yes''')
query = '''
select*
from BeforeLeap
order by NextWAR desc
'''

#convert the data into dataframe
df_beforeleap = pd.read_sql(query, sql_conn)

#take a look at the data
print(df_beforeleap.head())
```

```
   Season             Name       Team    G   PA  HR   R  RBI  SB    BBp  ...  \
0    2011     Buster Posey     Giants   45  185   4  17   21   3  0.097  ...
1    2003    Adrian Beltre    Dodgers  158  608  23  50   80   2  0.061  ...
2    2010  Jacoby Ellsbury    Red Sox   18   84   0  10    5   7  0.048  ...
3    2014     Bryce Harper  Nationals  100  395  13  41   32   2  0.096  ...
4    2008      Ben Zobrist       Rays   62  227  12  32   30   3  0.110  ...

   Pullp  Centp  Oppop  Softp   Medp  Hardp  WAR  NextWAR  jumpfromavg  jump
0  0.406  0.346  0.248  0.233  0.534  0.233  1.8     10.1         7.20   8.3
1  0.434  0.288  0.277  0.161  0.630  0.209  3.2      9.7         6.10   6.5
2  0.333  0.406  0.261  0.261  0.609  0.130 -0.2      9.5         8.55   9.7
3  0.389  0.353  0.258  0.179  0.520  0.302  1.6      9.3         6.45   7.7
4  0.491  0.307  0.203  0.129  0.558  0.313  1.5      8.7         8.50   7.2

[5 rows x 36 columns]
```

Step2: Import the data of the year that the player made the jump

In [2]:

```
#connect with sql server and retrieve the data we want
sql_conn = pyodbc.connect('''DRIVER={ODBC Driver 13 for SQL Server};
```

```
                              SERVER=ALLENHO\MSSQLSERVER002;
                              DATABASE=WAR Jump;
                              Trusted_Connection=yes''')
query = '''
select*
from LeapYear
order by WAR desc
'''

#convert the data into dataframe
df_leapyear = pd.read_sql(query, sql_conn)

#take a look at the data
print(df_leapyear.head())
```

```
   Season             Name       Team    G   PA  HR    R  RBI  SB    BBp  ...  \
0    2012     Buster Posey     Giants  148  610  24   78  103   1  0.113  ...
1    2004    Adrian Beltre    Dodgers  156  657  48  104  121   7  0.081  ...
2    2011  Jacoby Ellsbury    Red Sox  158  732  32  119  105  39  0.071  ...
3    2015     Bryce Harper  Nationals  153  654  42  118   99   6  0.190  ...
4    2009       Ben Zobrist       Rays  152  599  27   91   91  17  0.152  ...

   IFFBp   HRFB   IFHp  Pullp  Centp  Oppop  Softp   Medp  Hardp   WAR
0  0.039  0.188  0.083  0.382  0.363  0.255  0.108  0.576  0.316  10.1
1  0.145  0.240  0.052  0.361  0.305  0.334  0.132  0.528  0.340   9.7
2  0.104  0.167  0.091  0.397  0.353  0.251  0.240  0.495  0.265   9.5
3  0.058  0.273  0.046  0.454  0.338  0.208  0.119  0.472  0.409   9.3
4  0.052  0.175  0.066  0.488  0.306  0.207  0.124  0.557  0.318   8.7

[5 rows x 33 columns]
```

Step3: Print out the data of the two year differential and correlation table

In [5]:

```
df_beforeleaptrim=df_beforeleap.loc[:,'G':'WAR']
df_leapyeartrim=df_leapyear.loc[:,'G':'WAR']
df_diff=df_leapyeartrim-df_beforeleaptrim
df_corr_diff = df_diff.corr()
print(df_diff)
print(df_corr_diff.loc[:,'WAR'])
```

```
      G   PA  HR    R  RBI  SB    BBp     Kp    ISO  BABIP  ...  IFFBp   HRFB  \
0   103  425  20   61   82  -2  0.016 -0.005  0.108  0.042  ... -0.012  0.085
1    -2   49  25   54   41   5  0.020 -0.037  0.110  0.073  ... -0.007  0.105
2   140  648  32  109  100  32  0.023  0.027  0.179  0.119  ...  0.104  0.167
3    53  259  29   77   67   4  0.094 -0.063  0.168  0.017  ... -0.025  0.118
4    90  372  15   59   61  14  0.042  0.011 -0.007  0.074  ... -0.021  0.001
..  ...  ...  ..  ...  ...  ..    ...    ...    ...    ...  ...    ...    ...
58   43  279  11   37   42   7  0.015 -0.030  0.057  0.055  ...  0.048  0.032
59  107  349  11   53   39  16  0.049 -0.027 -0.036  0.133  ... -0.071  0.045
60   33  196  12   34   27   2 -0.003  0.058  0.065  0.042  ... -0.015  0.065
61   37  170  16   33   40   4  0.031 -0.002  0.110  0.070  ... -0.043  0.100
62   51  276  23   52   55  -3 -0.009  0.012  0.129  0.074  ...  0.067  0.110

     IFHp  Pullp  Centp  Oppop  Softp   Medp  Hardp  WAR
0   0.040 -0.024  0.017  0.007 -0.125  0.042  0.083  8.3
1  -0.040 -0.073  0.017  0.057 -0.029 -0.102  0.131  6.5
2   0.032  0.064 -0.053 -0.010 -0.021 -0.114  0.135  9.7
3  -0.039  0.065 -0.015 -0.050 -0.060 -0.048  0.107  7.7
4  -0.003 -0.003 -0.001  0.004 -0.005 -0.001  0.005  7.2
..    ...    ...    ...    ...    ...    ...    ...  ...
58  0.007  0.018  0.020 -0.038  0.042 -0.102  0.060  4.2
59  0.046  0.149  0.095 -0.244 -0.166  0.210 -0.044  4.0
60 -0.014 -0.031  0.055 -0.024 -0.082 -0.012  0.095  4.1
61 -0.048  0.080 -0.025 -0.056 -0.040 -0.057  0.097  4.7
62 -0.010 -0.075  0.051  0.026  0.014 -0.094  0.081  5.0

[63 rows x 30 columns]
G        0.180927
PA       0.182269
HR       0.133373
R        0.381860
RBI      0.242242
```

```
RBI       0.242242
SB        0.289972
BBp       0.117997
Kp        0.009606
ISO       0.132658
BABIP     0.307701
AVG       0.306928
OBP       0.325248
SLG       0.247819
wOBA      0.304482
wRCp      0.341955
BsR       0.241788
GBFB     -0.065691
LDp       0.139518
GBp      -0.108654
FBp       0.032023
IFFBp     0.181872
HRFB      0.052006
IFHp      0.031965
Pullp    -0.105371
Centp     0.037522
Oppop     0.098905
Softp    -0.016719
Medp     -0.031591
Hardp     0.044388
WAR       1.000000
Name: WAR, dtype: float64
```

Step4: Build a regression tree model and find out the MAE, RMSE, R2

In [6]:

```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import r2_score
import numpy as np


# split data into X and y
X = df_diff.loc[:,'G':'Hardp']
Y = df_diff.loc[:,'WAR']

# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)

# define the model
model = DecisionTreeRegressor(criterion = 'mse', max_depth=5)
model.fit(X_train, y_train)

# make predictions for test data
y_pred = model.predict(X_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R Squared Score is:', r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 1.7992673992673989
Root Mean Squared Error: 2.176697386477327
R Squared Score is: -1.362435668785717
```

The result was not a good one judging from negative R2 and so-so correlation coefficient, probably because I tried the correlation with WAR but taking into account of too much batting stats without enough fielding and running.

However still take a look at which stats contribute most in this model.

In [7]:

```python
for importance, name in sorted(zip(model.feature_importances_, X_train.columns),reverse=True):
 print(name, importance)
```

```
BsR 0.2687729434592346
RBI 0.24990735837065156
PA 0.11301931847193565
BABIP 0.10596875203372322
HR 0.06992560308600772
Medp 0.06808634810134878
IFHp 0.06668901419678447
Centp 0.02018452419689881
Pullp 0.017124139162415467
GBp 0.009157793385631252
GBFB 0.005615797553005234
SLG 0.004088300618584444
ISO 0.0014601073637788848
wRCp 0.0
wOBA 0.0
Softp 0.0
SB 0.0
R 0.0
Oppop 0.0
OBP 0.0
LDp 0.0
Kp 0.0
IFFBp 0.0
Hardp 0.0
HRFB 0.0
G 0.0
FBp 0.0
BBp 0.0
AVG 0.0
```

We turn to a new topic focusing on the batting stats, which include HR, R, BBp, Kp, ISO, BABIP, AVG, OBP, SLG, GBFB, LDp, GBp, FBp, IFFBp, HRFB, IFHp, Pullp, Centp, Oppop, Softp, Medp, Hardp and see if the differential of these categories betweem jump year and previous year are significantly correlated with the the differential of wRC+.

In [9]:

```python
df_beforeoff=df_beforeleap[['HR','R','BBp', 'Kp', 'ISO', 'BABIP', 'AVG', 'OBP', 'SLG', 'GBFB', 'LDp
', 'GBp', 'FBp', 'IFFBp', 'HRFB', 'IFHp', 'Pullp', 'Centp', 'Oppop', 'Softp', 'Medp', 'Hardp','wRCp
']]
df_leapoff=df_leapyear[['HR','R','BBp', 'Kp', 'ISO', 'BABIP', 'AVG', 'OBP', 'SLG', 'GBFB', 'LDp', '
GBp', 'FBp', 'IFFBp', 'HRFB', 'IFHp', 'Pullp', 'Centp', 'Oppop', 'Softp', 'Medp', 'Hardp','wRCp']]
df_diff_off=df_leapoff-df_beforeoff
df_corr_diff_off = df_diff_off.corr()
print(df_diff_off)
print(df_corr_diff_off.loc[:,'wRCp'])
```

```
    HR    R    BBp     Kp    ISO  BABIP    AVG    OBP    SLG  GBFB  ...  \
0   20   61  0.016 -0.005  0.108  0.042  0.052  0.040  0.160 -0.18  ...
1   25   54  0.020 -0.037  0.110  0.073  0.094  0.098  0.205 -0.07  ...
2   32  109  0.023  0.027  0.179  0.119  0.129  0.135  0.308 -0.16  ...
3   29   77  0.094 -0.063  0.168  0.017  0.057  0.116  0.226 -0.28  ...
4   15   59  0.042  0.011 -0.007  0.074  0.044  0.066  0.038  0.04  ...
..  ..  ...    ...    ...    ...    ...    ...    ...    ...   ...  ...
58  11   37  0.015 -0.030  0.057  0.055  0.059  0.072  0.115 -0.04  ...
59  11   53  0.049 -0.027 -0.036  0.133  0.115  0.141  0.079  0.67  ...
60  12   34 -0.003  0.058  0.065  0.042  0.019  0.015  0.084 -0.30  ...
61  16   33  0.031 -0.002  0.110  0.070  0.065  0.086  0.175 -0.20  ...
62  23   52 -0.009  0.012  0.129  0.074  0.073  0.072  0.202 -0.39  ...

    IFFBp   HRFB   IFHp  Pullp  Centp  Oppop  Softp   Medp  Hardp   wRCp
0  -0.012  0.085  0.040 -0.024  0.017  0.007 -0.125  0.042  0.083   48.0
1  -0.007  0.105 -0.040 -0.073  0.017  0.057 -0.029 -0.102  0.131   75.0
2   0.104  0.167  0.032  0.064 -0.053 -0.010 -0.021 -0.114  0.135  124.0
3  -0.025  0.118 -0.039  0.065 -0.015 -0.050 -0.060 -0.048  0.107   82.0
4  -0.021  0.001 -0.003 -0.003 -0.001  0.004 -0.005 -0.001  0.005   29.0
..    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
58  0.048  0.032  0.007  0.018  0.020 -0.038  0.042 -0.102  0.060   47.0
59 -0.071  0.045  0.046  0.149  0.095 -0.244 -0.166  0.210 -0.044   67.0
60 -0.015  0.065 -0.014 -0.031  0.055 -0.024 -0.082 -0.012  0.095   30.0
61 -0.043  0.100 -0.048  0.080 -0.025 -0.056 -0.040 -0.057  0.097   79.0
62  0.067  0.110 -0.010 -0.075  0.051  0.026  0.014 -0.094  0.081   71.0

[63 rows x 23 columns]
HR         0.508888
```

```
R         0.213211
BBp       0.227693
Kp        0.150763
ISO       0.703420
BABIP     0.371309
AVG       0.571028
OBP       0.761551
SLG       0.877834
GBFB     -0.325276
LDp       0.033460
GBp      -0.321055
FBp       0.282038
IFFBp     0.083147
HRFB      0.648738
IFHp     -0.132628
Pullp     0.283420
Centp    -0.092512
Oppop    -0.268495
Softp    -0.167955
Medp     -0.237407
Hardp     0.371935
wRCp      1.000000
Name: wRCp, dtype: float64
```

Visualize the correlation table

In [10]:

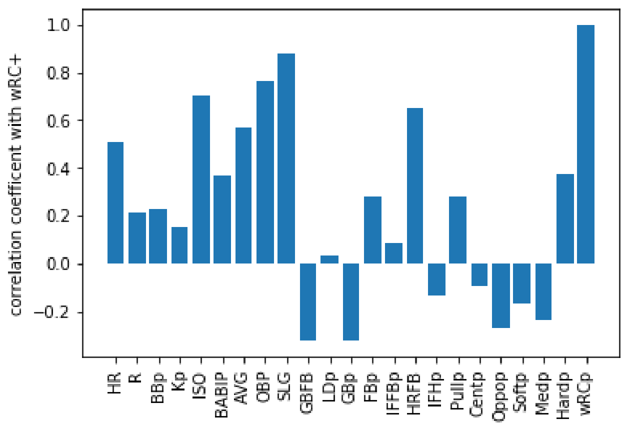
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress
fig, ax = plt.subplots()

#-----plot the correlation table as bar plot----------------------
ax.bar(df_corr_diff_off.index, df_corr_diff_off['wRCp'])

ax.set_xticklabels(df_corr_diff_off.index, rotation=90)

ax.set_ylabel('correlation coefficent with wRC+')

plt.show()
```



When taking a deep look into the data, we can see that SLG, OBP, ISO, HR, HR/FB are the categories that had more than 0.5 correlation coefficient. These gives us an idea that players making a huge offensive leap were more inclined through power surge

Also if we focus on the batted ball type(LDp, GBp, FBp), we can see that increased FBp is a good indicator of increased wRC+, which probably imply that of these players making a leap, having more Fly ball is largely responsible for it.

If we focus on Batted ball direction(Pullp, Centp, Oppop), Pullp is the only one that has positive correlation with wRC+.

If we focus on Quality of contact(Softp, Medp, Hardp), Hardp is expectedly the one.

Although we can not imply the player who has more Fly ball, Pull and Hard Contact percentage can have a leap year from this study, judging from the data of these already proven players, these stats may be largely responsible for their offensive leap that year.

Then I tried to build a model for this offensive part of data and see which categories are more responsible for the model

In [11]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import r2_score
import numpy as np


# split data into X and y
X2 = df_diff_off.loc[:,'HR':'Hardp']
Y2 = df_diff_off.loc[:,'wRCp']

# split data into train and test sets
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, Y2, test_size=0.2)

# define the model
model = DecisionTreeRegressor(criterion = 'mse', max_depth=5)
model.fit(X2_train, y2_train)

# make predictions for test data
y2_pred = model.predict(X2_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y2_test, y2_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y2_test, y2_pred)))
print('R Squared Score is:', r2_score(y2_test, y2_pred))
```

```
Mean Absolute Error: 10.132867132867132
Root Mean Squared Error: 14.134053056958235
R Squared Score is: 0.6183085059347685
```

The result was pretty nice because more than 60% of the variance for the dependent variable(wRC+ gain) was explained by these independent variables in the model.

In [12]:

```
for importance, name in sorted(zip(model.feature_importances_, X2_train.columns),reverse=True):
 print(name, importance)
```

```
SLG 0.7315124241028729
OBP 0.14239912282977266
BABIP 0.07110624992434106
LDp 0.014588509322298824
Kp 0.013695551088390606
BBp 0.012385389451557028
AVG 0.00684480257718867
IFFBp 0.0026436263503857053
FBp 0.0021207758055316623
R 0.0015227287778218713
IFHp 0.0006344703240925692
HR 0.0002819868107078086
Centp 0.00021149010803085642
HRFB 5.287252700774616e-05
Softp 0.0
Pullp 0.0
Oppop 0.0
Medp 0.0
ISO 0.0
Hardp 0.0
GBp 0.0
GBFB 0.0
```

As it turned out it is the SLG, OBP that took the large component of responsibility of this model, which is not surprising given the fact that these two categories are popular and largely seen as a standard for players' offensive output.

The result between batting stats and wRC+ was pretty nice. Thus I used the same batting stats again, this time see the correlation with wOBA

In [13]:

```
df_beforeoff2=df_beforeleap[['HR','R','BBp', 'Kp', 'ISO', 'BABIP', 'AVG', 'OBP', 'SLG', 'GBFB', 'LD
p', 'GBp', 'FBp', 'IFFBp', 'HRFB', 'IFHp', 'Pullp', 'Centp', 'Oppop', 'Softp', 'Medp', 'Hardp','wOB
A']]
df_leapoff2=df_leapyear[['HR','R','BBp', 'Kp', 'ISO', 'BABIP', 'AVG', 'OBP', 'SLG', 'GBFB', 'LDp',
'GBp', 'FBp', 'IFFBp', 'HRFB', 'IFHp', 'Pullp', 'Centp', 'Oppop', 'Softp', 'Medp', 'Hardp','wOBA']]
df_diff_off2=df_leapoff2-df_beforeoff2
df_corr_diff_off2 = df_diff_off2.corr()
print(df_diff_off2)
print(df_corr_diff_off2.loc[:,'wOBA'])
```

```
    HR    R    BBp     Kp    ISO  BABIP    AVG    OBP    SLG  GBFB  ...  \
0   20   61  0.016 -0.005  0.108  0.042  0.052  0.040  0.160 -0.18  ...
1   25   54  0.020 -0.037  0.110  0.073  0.094  0.098  0.205 -0.07  ...
2   32  109  0.023  0.027  0.179  0.119  0.129  0.135  0.308 -0.16  ...
3   29   77  0.094 -0.063  0.168  0.017  0.057  0.116  0.226 -0.28  ...
4   15   59  0.042  0.011 -0.007  0.074  0.044  0.066  0.038  0.04  ...
..  ..  ...    ...    ...    ...    ...    ...    ...    ...   ...  ...
58  11   37  0.015 -0.030  0.057  0.055  0.059  0.072  0.115 -0.04  ...
59  11   53  0.049 -0.027 -0.036  0.133  0.115  0.141  0.079  0.67  ...
60  12   34 -0.003  0.058  0.065  0.042  0.019  0.015  0.084 -0.30  ...
61  16   33  0.031 -0.002  0.110  0.070  0.065  0.086  0.175 -0.20  ...
62  23   52 -0.009  0.012  0.129  0.074  0.073  0.072  0.202 -0.39  ...

    IFFBp   HRFB   IFHp  Pullp  Centp  Oppop  Softp   Medp  Hardp   wOBA
0  -0.012  0.085  0.040 -0.024  0.017  0.007 -0.125  0.042  0.083  0.071
1  -0.007  0.105 -0.040 -0.073  0.017  0.057 -0.029 -0.102  0.131  0.118
2   0.104  0.167  0.032  0.064 -0.053 -0.010 -0.021 -0.114  0.135  0.178
3  -0.025  0.118 -0.039  0.065 -0.015 -0.050 -0.060 -0.048  0.107  0.123
4  -0.021  0.001 -0.003 -0.003 -0.001  0.004 -0.005 -0.001  0.005  0.045
..    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
58  0.048  0.032  0.007  0.018  0.020 -0.038  0.042 -0.102  0.060  0.072
59 -0.071  0.045  0.046  0.149  0.095 -0.244 -0.166  0.210 -0.044  0.099
60 -0.015  0.065 -0.014 -0.031  0.055 -0.024 -0.082 -0.012  0.095  0.041
61 -0.043  0.100 -0.048  0.080 -0.025 -0.056 -0.040 -0.057  0.097  0.107
62  0.067  0.110 -0.010 -0.075  0.051  0.026  0.014 -0.094  0.081  0.106

[63 rows x 23 columns]
HR        0.573070
R         0.209447
BBp       0.284463
Kp        0.180227
ISO       0.747745
BABIP     0.342174
AVG       0.550492
OBP       0.784717
SLG       0.909761
GBFB     -0.346554
LDp       0.006658
GBp      -0.340681
FBp       0.313535
IFFBp     0.047101
HRFB      0.682351
IFHp     -0.144632
Pullp     0.290097
Centp    -0.111656
Oppop    -0.262779
Softp    -0.184035
Medp     -0.250701
Hardp     0.396958
wOBA      1.000000
Name: wOBA, dtype: float64
```

In [15]:

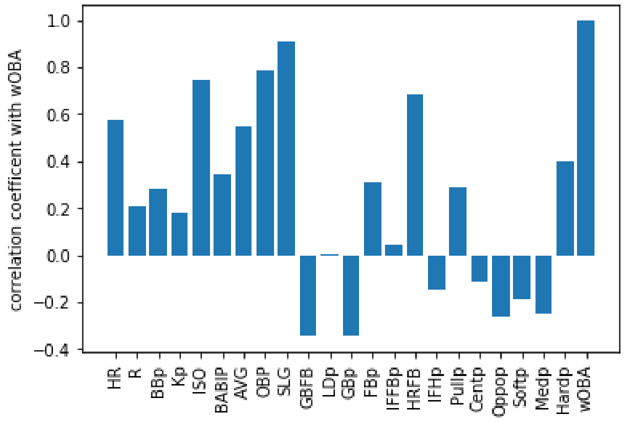
```
fig, ax = plt.subplots()

#-----plot the correlation table as bar plot----------------------
ax.bar(df_corr_diff_off2.index, df_corr_diff_off2['wOBA'])

ax.set_xticklabels(df_corr_diff_off2.index, rotation=90)

ax.set_ylabel('correlation coefficent with wOBA')

plt.show()
```



In [17]:

```
# split data into X and y
X3 = df_diff_off2.loc[:,'HR':'Hardp']
Y3 = df_diff_off2.loc[:,'wOBA']

# split data into train and test sets
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, Y3, test_size=0.2)

# define the model
model = DecisionTreeRegressor(criterion = 'mse', max_depth=5)
model.fit(X3_train, y3_train)

# make predictions for test data
y3_pred = model.predict(X3_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y3_test, y3_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y3_test, y3_pred)))
print('R Squared Score is:', r2_score(y3_test, y3_pred))
```

```
Mean Absolute Error: 0.016232142857142858
Root Mean Squared Error: 0.020314841621258288
R Squared Score is: 0.7484161489396279
```

The result between batting stats and wOBA was even better. More than 74% of the variance for the dependent variable(wOBA) was explained by these independent variables in the model. MAE and RMSE were smaller in this case as well.

In [18]:

```
for importance, name in sorted(zip(model.feature_importances_, X3_train.columns),reverse=True):
 print(name, importance)
```

```
SLG 0.8505594411723238
OBP 0.060847303016411745
LDp 0.03592799110241595
HRFB 0.02082987334541791
Centp 0.019799483844447755
IFFBp 0.005093637431345794
AVG 0.0019041223323669565
Kp 0.0017048317509533082
IFHp 0.001410460986938324
BBp 0.0009944851880561482
Oppop 0.0006308507148603866
Hardp 0.0002975191144319763
Softp 0.0
R 0.0
Pullp 0.0
Medp 0.0
ISO 0.0
HR 0.0
GBp 0.0
GBFB 0.0
FBp 0.0
BABIP 0.0
```

When again taking a deep look into the data, we can see that HR, ISO, AVG, OBP, SLG, HR/FB are still the categories that had more than 0.5 correlation coefficient. If we focus on the batted ball type(LDp, GBp, FBp), Batted ball direction(Pullp, Centp, Oppop), Quality of contact(Softp, Medp, Hardp), more fly ball, pull and hard contact percentage are still the main conclusion, further varify the result.

Conclusion

From this study we found out that the categories pushing players making a leap(four war differential from last year) were power related(SLG, OBP, ISO, HR, HR/FB) and more fly ball, pull and hard contact percentage. Although we should not make more assumption other than this from this study alone, this research may give us an idea of how player can approach in order for a potential offensive leap.