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;
```

```
                                 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 [3]:

```
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 [4]:

```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: 1.41965811965812
Root Mean Squared Error: 1.9053159003068272
R Squared Score is: -3.1280355733654996
```

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 [5]:

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

```
RBI 0.2639820681007306
BsR 0.20884883897616013
FBp 0.15727920725293487
IFHp 0.08028381181378229
Hardp 0.07342123486729539
OBP 0.05461083585170881
Centp 0.0455564351158791
Pullp 0.038834372161215776
R 0.030436872600683968
Oppop 0.018582853866206672
wRCp 0.010267974865867623
IFFBp 0.008755764360685573
BBp 0.0068453165772456755
GBp 0.002294413589603433
wOBA 0.0
Softp 0.0
SLG 0.0
SB 0.0
PA 0.0
Medp 0.0
LDp 0.0
Kp 0.0
ISO 0.0
HRFB 0.0
HR 0.0
GBFB 0.0
G 0.0
BABIP 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 [6]:

```
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 [7]:

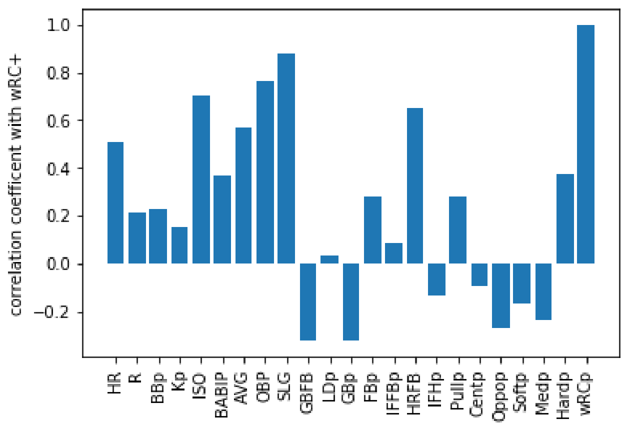
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress
fig, ax = plt.subplots()

#-----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 [8]:

```
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: 10.493589743589743
Root Mean Squared Error: 13.593829513963342
R Squared Score is: 0.520852405037828
```

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 [9]:

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

```
SLG 0.7888424897285162
BABIP 0.07756003465219917
OBP 0.04422864894799582
AVG 0.036330786446085485
ISO 0.009610696079746519
Hardp 0.00955374954024159
HR 0.008283865404097404
Medp 0.0074212161607234714
Softp 0.007059712261536189
IFFBp 0.0049759112188758284
BBp 0.003012993225747101
Centp 0.0025227869879699606
GBp 0.0005971093462650993
R 0.0
Pullp 0.0
Oppop 0.0
LDp 0.0
Kp 0.0
IFHp 0.0
HRFB 0.0
GBFB 0.0
FBp 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 [10]:

```python
df_beforeoff2=df_beforeleap[['HR','R','BBp', 'Kp', 'ISO', 'BABIP', 'AVG', 'OBP', 'SLG', 'GBFB', 'LDp', 'GBp', 'FBp', 'IFFBp', 'HRFB', 'IFHp', 'Pullp', 'Centp', 'Oppop', 'Softp', 'Medp', 'Hardp','wOBA']]
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 [11]:

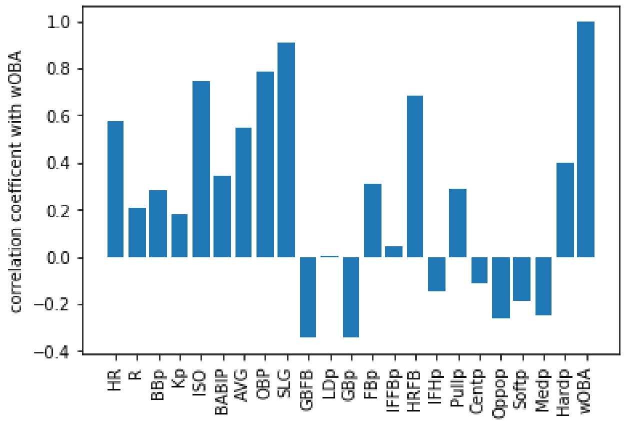
```python
fig, ax = plt.subplots()

#-----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 [12]:

```
# 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: 0.021423076923076 92
Root Mean Squared Error: 0.028001564364113787
R Squared Score is: 0.6240678904820174
```

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 [13]:

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

```
SLG 0.8466092959253388
OBP 0.06540956078790922
Hardp 0.03745860765312268
Kp 0.01620633655898944
IFFBp 0.006960488972032294
AVG 0.006242448940322371 5
BABIP 0.006010343995637105
Medp 0.0038981576658499244
Centp 0.003801448808169726
BBp 0.002595021014409118
GBp 0.002129028558221297
Softp 0.0011039407338925063
FBp 0.001097858415799465
Pullp 0.00032844517702587304
Oppop 0.00014901679328022504
R 0.0
LDp 0.0
ISO 0.0
IFHp 0.0
HRFB 0.0
HR 0.0
GBFB 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.

After I distinguished which categories may contributed most to the players' war jump. I decided to see if I can categorize these players based on what they did in their leap year by using clustering model.

In [14]:

```
print(df_leapyear.loc[:,'G':'WAR'].describe())
```

```
                G          PA         HR           R         RBI          SB  \
count   63.000000   63.000000  63.000000   63.000000   63.000000   63.000000
mean   146.126984  613.365079  26.634921   92.095238   87.111111   14.206349
std     12.969557   87.268274  11.912722   19.548710   23.071496   14.310112
min    103.000000  329.000000   3.000000   27.000000   35.000000    0.000000
25%    140.000000  569.500000  20.000000   83.000000   73.000000    4.500000
50%    148.000000  635.000000  25.000000   94.000000   85.000000   10.000000
75%    157.000000  680.000000  31.500000  102.500000  101.500000   17.000000
max    161.000000  754.000000  59.000000  129.000000  139.000000   64.000000

             BBp         Kp        ISO      BABIP  ...      IFFBp       HRFB  \
count  63.000000  63.000000  63.000000  63.000000  ...  63.000000  63.000000
mean    0.104413   0.180587   0.230460   0.334651  ...   0.081016   0.168667
std     0.035691   0.050872   0.060688   0.031102  ...   0.038324   0.068702
min     0.040000   0.075000   0.099000   0.233000  ...   0.006000   0.032000
25%     0.079500   0.139000   0.196500   0.315500  ...   0.049000   0.125500
50%     0.101000   0.174000   0.222000   0.336000  ...   0.081000   0.167000
75%     0.122000   0.225000   0.263000   0.358500  ...   0.102500   0.206000
max     0.206000   0.296000   0.359000   0.393000  ...   0.169000   0.343000

            IFHp      Pullp      Centp      Oppop      Softp       Medp  \
count  63.000000  63.000000  63.000000  63.000000  63.000000  63.000000
mean    0.067603   0.421143   0.330794   0.248302   0.145762   0.513381
std     0.028647   0.064708   0.036965   0.048644   0.029443   0.058418
min     0.013000   0.314000   0.208000   0.137000   0.081000   0.376000
25%     0.044500   0.367000   0.305500   0.212000   0.127000   0.486000
50%     0.066000   0.408000   0.330000   0.246000   0.144000   0.517000
75%     0.088000   0.471500   0.359000   0.282500   0.164500   0.545500
max     0.130000   0.578000   0.407000   0.356000   0.240000   0.690000

           Hardp        WAR
count  63.000000  63.000000
mean    0.341016   6.131746
std     0.059442   1.634983
min     0.196000   3.500000
25%     0.314000   4.700000
50%     0.338000   5.900000
75%     0.374000   7.200000
max     0.492000  10.100000

[8 rows x 30 columns]
```

scale the numeric variables first

In [16]:

```
from sklearn.preprocessing import StandardScaler

df_leapyear_reserve = df_leapyear.loc[:, 'G':'WAR']

# scale and center numeric columns
to_scale = ['G','PA','HR','R','RBI','SB','BBp','Kp','ISO','BABIP','AVG','OBP','SLG','wOBA','wRCp','
BsR','GBFB','LDp','GBp','FBp',
'IFFBp','HRFB','IFHp','Pullp','Centp','Oppop','Softp','Medp','Hardp','WAR']

# scale and center numeric columns
df_leapyear[to_scale] = StandardScaler().fit_transform(df_leapyear[to_scale])
```

Perform PCA analysis

In [17]:

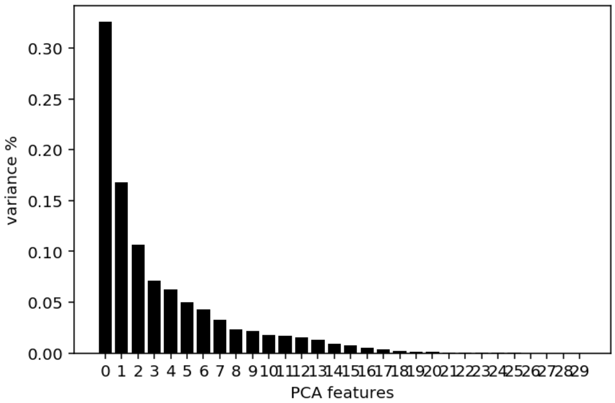
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
%config InlineBackend.figure_format='retina'

X = df_leapyear.loc[:, 'G':'WAR']

# Create a PCA instance: pca
pca = PCA(n_components=30)
principalComponents = pca.fit_transform(X)

# Plot the explained variances
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='black')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)

# Save components to a DataFrame
PCA_components = pd.DataFrame(principalComponents)
```

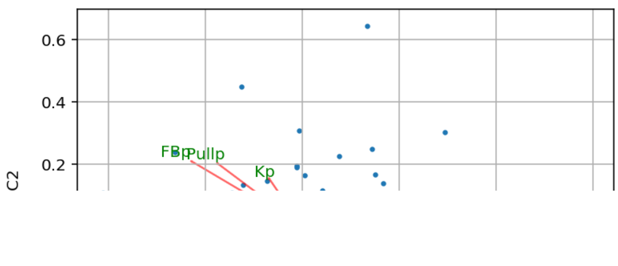


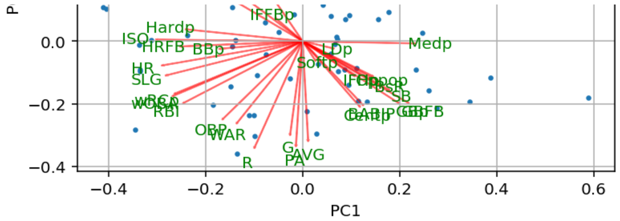
In [18]:

```
def myplot(score,coeff,labels=None):
    xs = score[:,0]
    ys = score[:,1]
    n = coeff.shape[0]
    scalex = 1.0/(xs.max() - xs.min())
    scaley = 1.0/(ys.max() - ys.min())
    plt.scatter(xs * scalex,ys * scaley,s=5)
    for i in range(n):
        plt.arrow(0, 0, coeff[i,0], coeff[i,1],color = 'r',alpha = 0.5)
        if labels is None:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = 'green', ha = 'cen
ter', va = 'center')
        else:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, labels[i], color = 'g', ha = 'center', va
= 'center')

    plt.xlabel("PC{}".format(1))
    plt.ylabel("PC{}".format(2))
    plt.grid()

myplot(principalComponents[:,0:2],np.transpose(pca.components_[0:2, :]),list(X.columns))
plt.show()
```





Result wasn't quite satisfying for lack of clear clusters

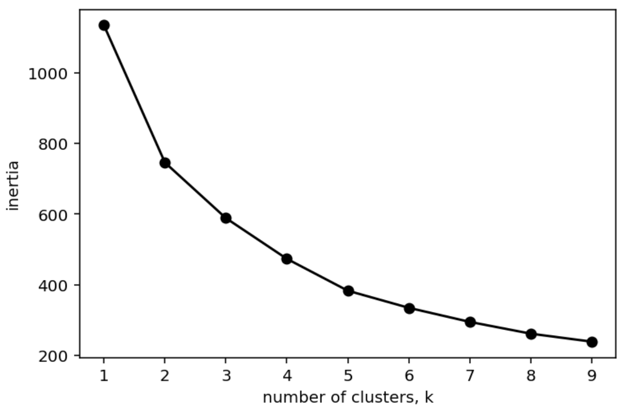
In [19]:

```
ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(PCA_components.iloc[:,:3])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```



According to the elbow plot, the best clustering number should be two or three. Two might be slightly better.

In [20]:

```
kmeans = KMeans(n_clusters=2).fit(X)
labels = pd.DataFrame(kmeans.labels_)
labeledPlayers = pd.concat((df_leapyear_reserve, labels),axis=1)
labeledPlayers = labeledPlayers.rename({0:'labels'},axis=1)
print(labeledPlayers.head())
```

	G	PA	HR	R	RBI	SB	BBp	Kp	ISO	BABIP	...	HRFB	IFHp	\
0	148	610	24	78	103	1	0.113	0.157	0.213	0.368	...	0.188	0.083	
1	156	657	48	104	121	7	0.081	0.132	0.294	0.326	...	0.240	0.052	
2	158	732	32	119	105	39	0.071	0.134	0.230	0.336	...	0.167	0.091	
3	153	654	42	118	99	6	0.190	0.200	0.319	0.369	...	0.273	0.046	
4	152	599	27	91	91	17	0.152	0.174	0.246	0.326	...	0.175	0.066	

	Pullp	Centp	Oppop	Softp	Medp	Hardp	WAR	labels
0	0.382	0.363	0.255	0.108	0.576	0.316	10.1	0
1	0.361	0.305	0.334	0.132	0.528	0.340	9.7	1
2	0.397	0.353	0.251	0.240	0.495	0.265	9.5	0
3	0.454	0.338	0.208	0.119	0.472	0.409	9.3	1
4	0.488	0.306	0.207	0.124	0.557	0.318	8.7	1

[5 rows x 31 columns]

Take a look at how the two clusters distribute in every category

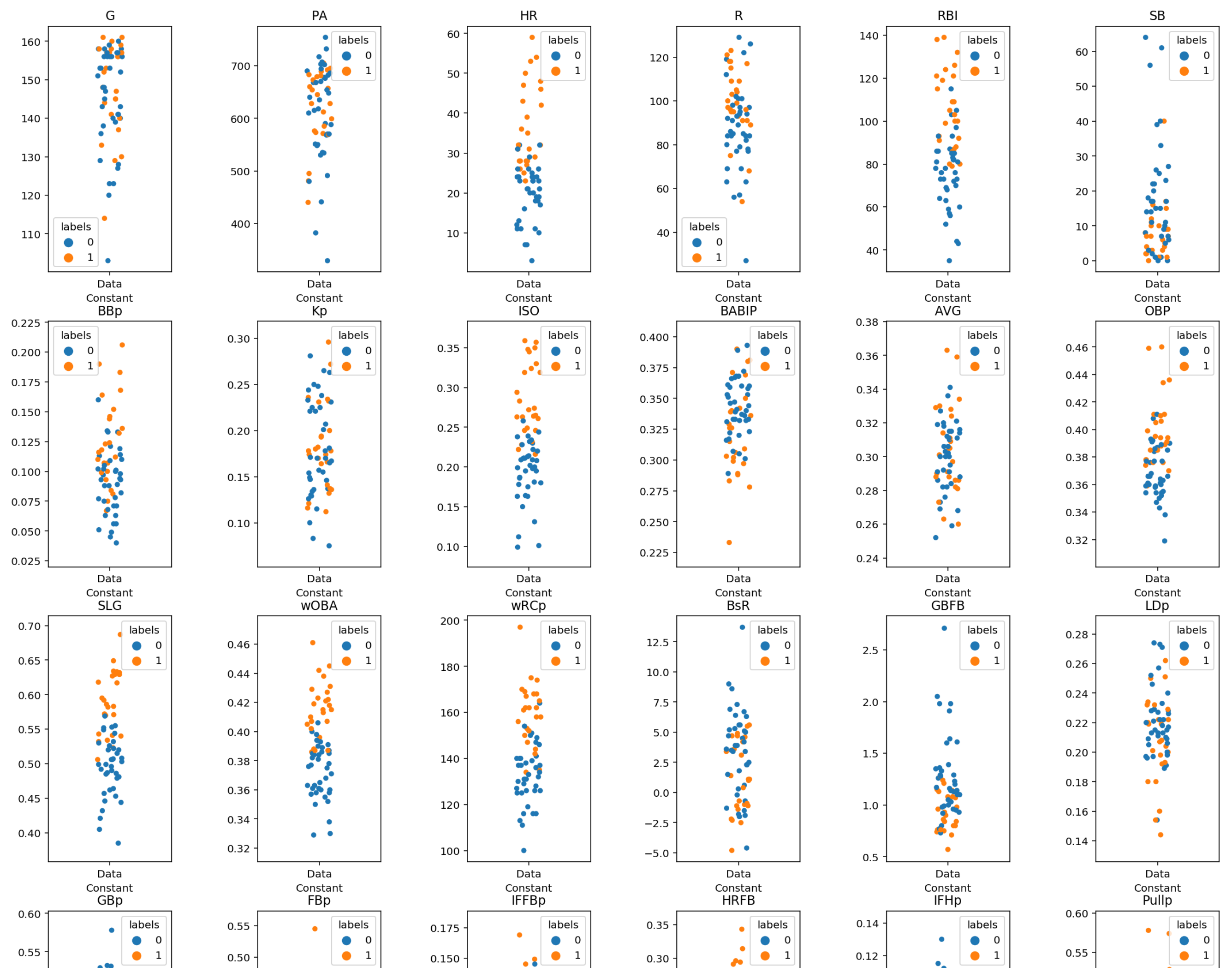
In [28]:

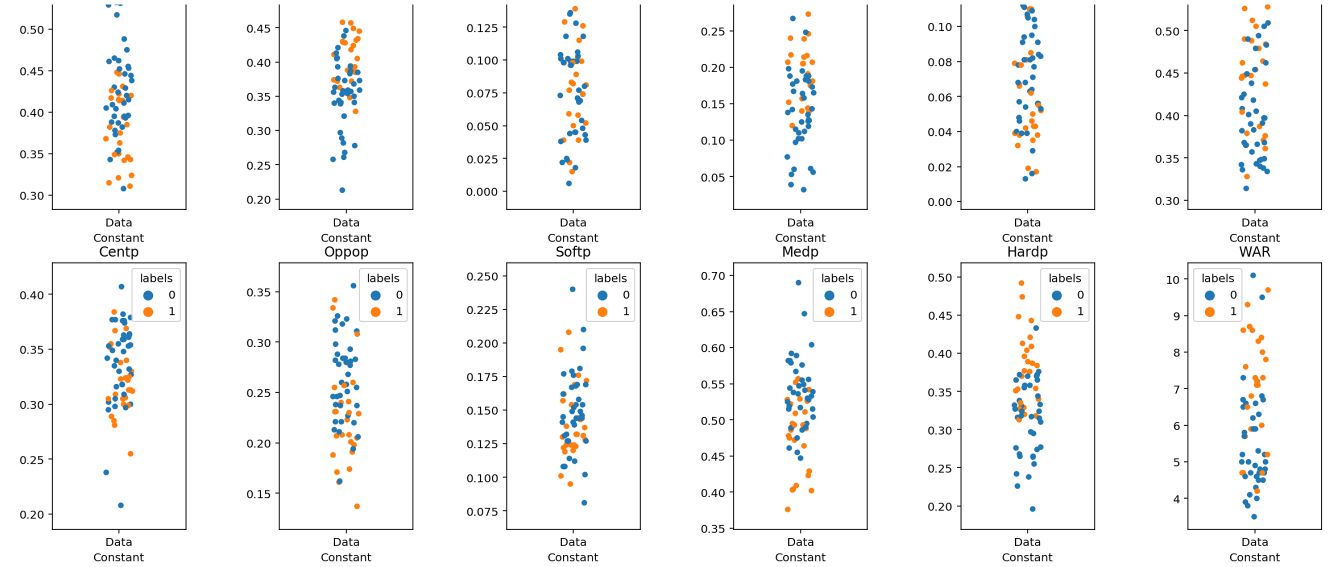
```
import seaborn as sns

labeledPlayers['Constant'] = "Data"
labeledPlayersx=labeledPlayers.loc[:,'G':'Constant']

f, axes = plt.subplots(5, 6, figsize=(20, 25), sharex=False)
f.subplots_adjust(hspace=0.2, wspace=0.7)

for i in range(0,30):
    col = labeledPlayersx.columns[i]
    if i < 6:
        ax = sns.stripplot(x=labeledPlayersx['Constant'],y=labeledPlayersx[col].values,hue=labeledP
layersx['labels'],jitter=True,ax=axes[0,(i)])
        ax.set_title(col)
    elif i >= 6 and i < 12:
        ax = sns.stripplot(x=labeledPlayersx['Constant'],y=labeledPlayersx[col].values,hue=labeledP
layersx['labels'],jitter=True,ax=axes[1,(i-6)])
        ax.set_title(col)
    elif i >= 12 and i < 18:
        ax = sns.stripplot(x=labeledPlayersx['Constant'],y=labeledPlayersx[col].values,hue=labeledP
layersx['labels'],jitter=True,ax=axes[2,(i-12)])
        ax.set_title(col)
    elif i >= 18 and i < 24:
        ax = sns.stripplot(x=labeledPlayersx['Constant'],y=labeledPlayersx[col].values,hue=labeledP
layersx['labels'],jitter=True,ax=axes[3,(i-18)])
        ax.set_title(col)
    elif i >= 24 and i < 30:
        ax = sns.stripplot(x=labeledPlayersx['Constant'],y=labeledPlayersx[col].values,hue=labeledP
layersx['labels'],jitter=True,ax=axes[4,(i-24)])
        ax.set_title(col)
```





As it turned out, if we tried to categorize these leaping players, ISO, SLG, wOBA, wRC+ were the ones that made more difference in the model.

In [29]:

```
ClusterTable = pd.concat((df_leapyear['Name'], labels),axis=1)
ClusterTable = ClusterTable.rename({0:'labels'},axis=1)
print(ClusterTable)
```

```
                 Name  labels
0        Buster Posey       0
1      Adrian Beltre        1
2     Jacoby Ellsbury       0
3        Bryce Harper       1
4         Ben Zobrist       1
..                ...     ...
58     Logan Forsythe       0
59       Jeff DaVanon       0
60       Jason Castro       0
61      J.D. Martinez       0
62       Jose Guillen       0

[63 rows x 2 columns]
```

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. As for the categorization of these players based on what they did in their leap year, ISO, SLG, wOBA, wRC+ were the ones that made more difference in the model.