MoneyBall

August 13, 2015

1 Machine Learning for the MoneyBall Predictions

data comes from BaseballReference.com

taken from EdX course: "The Analytics Edge" Can we discover what the Oakland A's need to make it to the playoffs in 2002?

The Statistician behind the decisions that year was -now famous- Paul DePodesta. We want to see if we can use Linear Regression to duplicate his findings and his recommendations to the team manager and owner that season.

In [2]:
```
%pylab inline
%matplotlib inline

from __future__ import division
from sklearn import linear_model
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

baseball = pd.read_csv("baseball.csv")

moneyball = baseball[baseball["Year"] < 2002]
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [3]:
```
baseball[:15]
```

Out[3]:

	Team	League	Year	RS	RA	W	OBP	SLG	BA	Playoffs	RankSeason \
0	ARI	NL	2012	734	688	81	0.328	0.418	0.259	0	NaN
1	ATL	NL	2012	700	600	94	0.320	0.389	0.247	1	4
2	BAL	AL	2012	712	705	93	0.311	0.417	0.247	1	5
3	BOS	AL	2012	734	806	69	0.315	0.415	0.260	0	NaN
4	CHC	NL	2012	613	759	61	0.302	0.378	0.240	0	NaN
5	CHW	AL	2012	748	676	85	0.318	0.422	0.255	0	NaN
6	CIN	NL	2012	669	588	97	0.315	0.411	0.251	1	2
7	CLE	AL	2012	667	845	68	0.324	0.381	0.251	0	NaN
8	COL	NL	2012	758	890	64	0.330	0.436	0.274	0	NaN
9	DET	AL	2012	726	670	88	0.335	0.422	0.268	1	6
10	HOU	NL	2012	583	794	55	0.302	0.371	0.236	0	NaN
11	KCR	AL	2012	676	746	72	0.317	0.400	0.265	0	NaN
12	LAA	AL	2012	767	699	89	0.332	0.433	0.274	0	NaN
13	LAD	NL	2012	637	597	86	0.317	0.374	0.252	0	NaN
14	MIA	NL	2012	609	724	69	0.308	0.382	0.244	0	NaN

```
    RankPlayoffs    G   OOBP   OSLG
0            NaN  162  0.317  0.415
1              5  162  0.306  0.378
2              4  162  0.315  0.403
3            NaN  162  0.331  0.428
4            NaN  162  0.335  0.424
5            NaN  162  0.319  0.405
6              4  162  0.305  0.390
7            NaN  162  0.336  0.430
8            NaN  162  0.357  0.470
9              2  162  0.314  0.402
10           NaN  162  0.337  0.427
11           NaN  162  0.339  0.423
12           NaN  162  0.310  0.403
13           NaN  162  0.310  0.364
14           NaN  162  0.327  0.399
```

The data has a little more than a thousand rows, and represents statistics from over 40 teams since 1962 until more recently. We are only trying to confirm DePodesta's predictions, so we will only use the data previous to 2002.

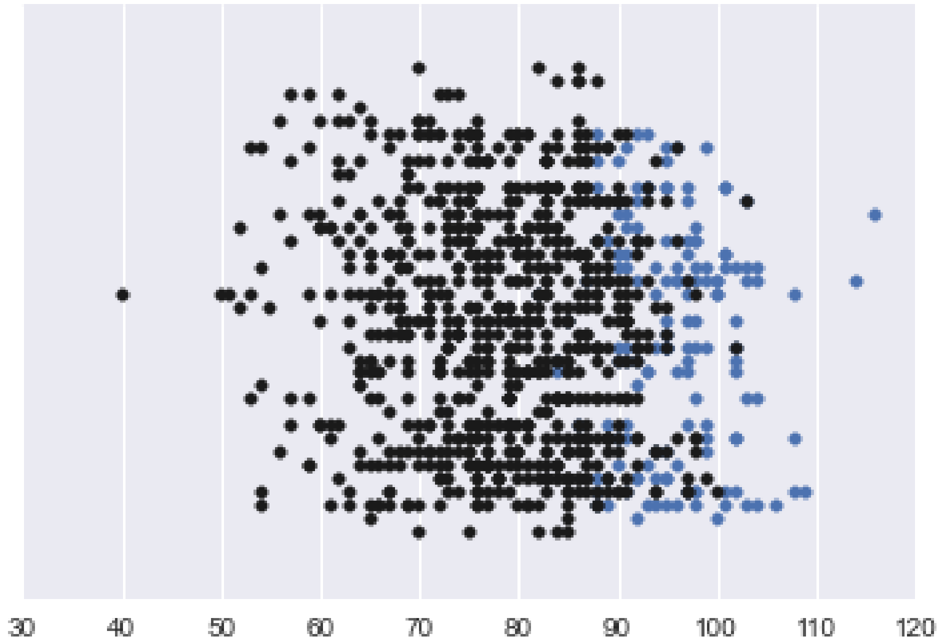
1.1 We plot the number of wins for each team – adding a third dimension (color) which encodes whether the team went to the playoffs or not.

We first find out when a team went to the playoffs and the number of wins they had that season.

```
In [14]: team_idx = {v:k for (k, v) in enumerate(moneyball["Team"].unique())}
         moneyball.loc[:,"_TeamID"] = [team_idx[x] for x in moneyball["Team"]]
```

```
In [5]: moneyball_playoff = moneyball[moneyball["Playoffs"]==1]
        plt.scatter(moneyball_playoff["W"], moneyball_playoff["_TeamID"], color="b")
        moneyball_nonplayoff = moneyball[moneyball["Playoffs"]==0]
        plt.scatter(moneyball_nonplayoff["W"], moneyball_nonplayoff["_TeamID"], color="k")
        plt.yticks([]) #-- just gets rid of y labels --#
```

```
Out[5]: ([], <a list of 0 Text yticklabel objects>)
```

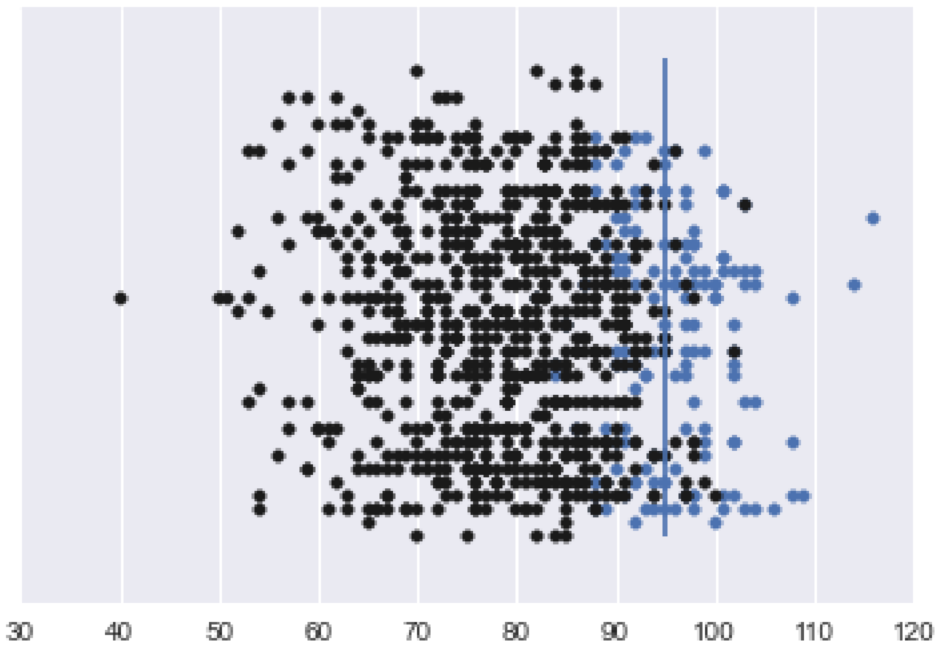


2 DePodesta estimated that a team needed 95 wins in order to reach the playoffs... which is in accordance, fairly well, with our scatterplot:

We mark a vertical line at 95 wins, below:

```
In [6]: plt.scatter(moneyball_playoff["W"], moneyball_playoff["_TeamID"], color="b")
        plt.scatter(moneyball_nonplayoff["W"], moneyball_nonplayoff["_TeamID"], color="k")
        plt.vlines(95, 0, len(team_idx), colors="b", linestyles="solid")
        plt.yticks([])
```

```
Out[6]: ([], <a list of 0 Text yticklabel objects>)
```



3 So... if Wins is a good predictor of making the playoffs (duh?!)... what makes a good predictor of Wins?

3.1 -DePodesta also estimated that to make those 95 Wins in the season, a team would need to score about 135 more runs than their opposing team.

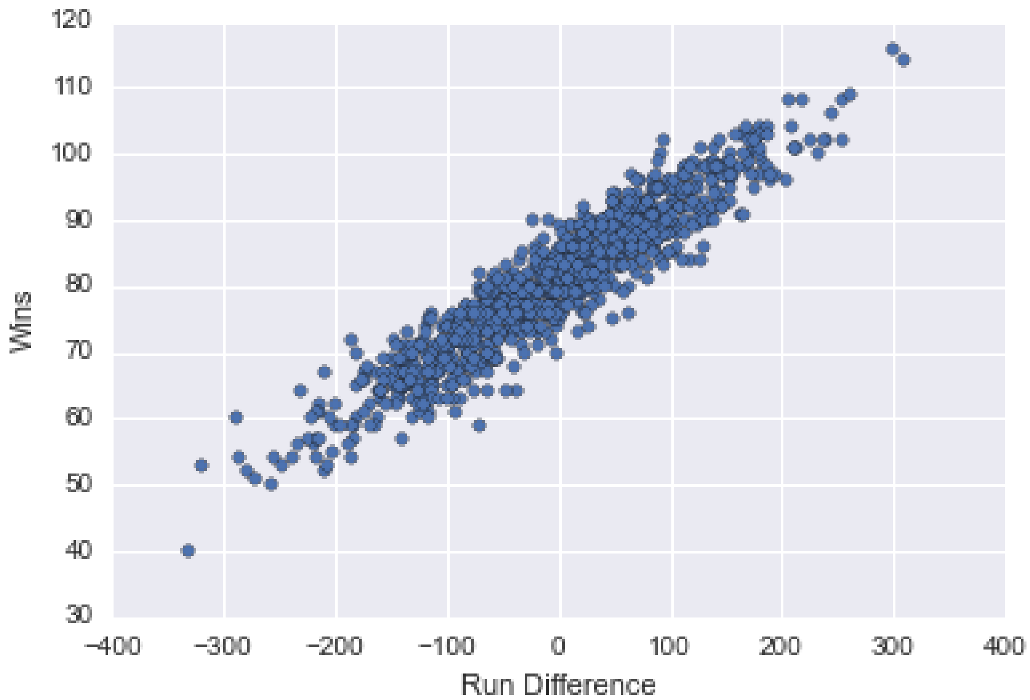
3.1.1 We use two other variables in our dataset:

Runs Against, RA
-and-
Runs Scored, RS

3.2 We can see how well this theory of DePodesta holds up. Will scoring more than 135 runs than their opponent lead to the playoffs?

Thusly, we plot this: Number of Wins against Difference of Runs Against/Scored (RS-RA)

```
In [7]: moneyball.loc[:,"_RDiff"] = moneyball["RS"] - moneyball["RA"]
        plt.scatter(moneyball["_RDiff"], moneyball["W"])
        plt.xlabel("Run Difference")
        plt.ylabel("Wins")

Out[7]: <matplotlib.text.Text at 0x109f7ec10>
```



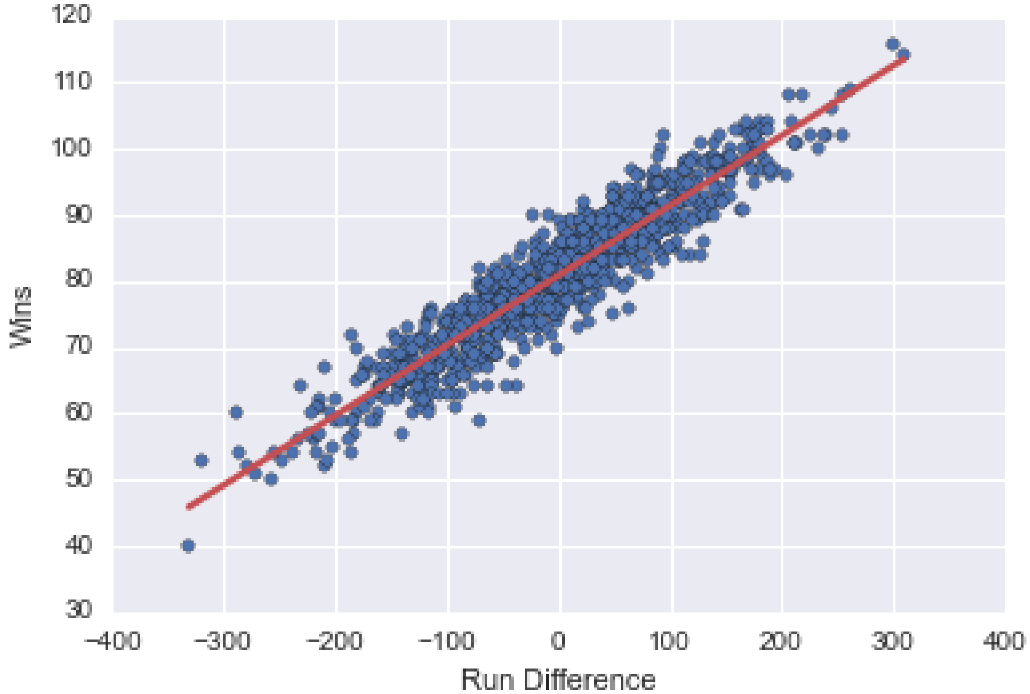
3.3 And we DEFINITELY see a pretty high correlation!!

-We DO have that Run Difference is a strong predictor of Wins. (This also makes intuitive sense.)
But... how good of a predictor is it?

3.4 We run a linear regression model to see how good the fit is to the data:

In [8]:
```
reg = linear_model.LinearRegression()
reg.fit(np.matrix(moneyball["_RDiff"]).T, moneyball["W"])

plt.scatter(moneyball["_RDiff"], moneyball["W"])
plt.xlabel("Run Difference")
plt.ylabel("Wins")

xs = [np.min(moneyball["_RDiff"]), np.max(moneyball["_RDiff"])]
ys = [reg.predict(x) for x in xs]
plt.plot(xs, ys, 'r', linewidth=2.5)
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x105baa410>]
```



3.5 And we find the following R^2 value:

```
In [9]: print("R-squared[RS_1]: %.4f" % (reg.score(np.matrix(moneyball["_RDiff"]).T, moneyball["W"])))
```

```
R-squared[RS_1]: 0.8808
```

3.6 And what Run Difference would this linear regression model predict would be needed to achieve 95 wins?

```
In [10]: run_diff_to_win = (95 - reg.intercept_) / reg.coef_[0]
         print("Run difference: %.2f" % (run_diff_to_win))
```

```
Run difference: 133.49
```

3.7 Just as DePodesta had estimated: a team needs 135 more runs than their opponents to make 95 Wins in the season (which would then lead to a playoff entry)

4 OK... "Score More Runs!" you tell me. We already could have guessed that.

4.1 How else can the Data show us how to improve our success??

Well... we need to break down what is a good predictor of Runs Scored (RS) and Runs Against (RA)... so let's keep doing just that!

4.1.1 What are good predictors of RS/RA?

4.1.2 NOTE: one of the assertions of the MoneyBall team was that statistical models that were predictors of Runs Against and Runs Scored would outperform human scouts!

5 Now... let's fit a model between Runs Scored (RS) and some of our other variables in the data set:

-On Base Percentage (OBP)
-Slugging Percentage (SLG)
-Batting Average (BA)

5.0.3 After fitting a linear model to these variables, we find that we have a very good fit

($R^2 = 0.93$)

```
In [11]: y = np.array(moneyball["RS"])
         X = np.vstack((np.array(moneyball["OBP"]),
                        np.array(moneyball["SLG"]),
                        np.array(moneyball["BA"]))).T
         rs_model = linear_model.LinearRegression()
         rs_model.fit(X, y)
         print("R-squared[RS_1]: %.4f" % (rs_model.score(X, y)))
         print"\n"
         print("Intercept: %.4f " % rs_model.intercept_)
         print("On Base Percentage Coefficient: %.2f" % rs_model.coef_[0])
         print("Slugging Percentage Coefficient: %.2f" % rs_model.coef_[1])
         print("Batting Average Coefficient: %.2f" % rs_model.coef_[2])
```

```
R-squared[RS_1]: 0.9302


Intercept: -788.4570
On Base Percentage Coefficient: 2917.42
Slugging Percentage Coefficient: 1637.93
Batting Average Coefficient: -368.97
```

5.1 However... look closely at the estimated coefficients:

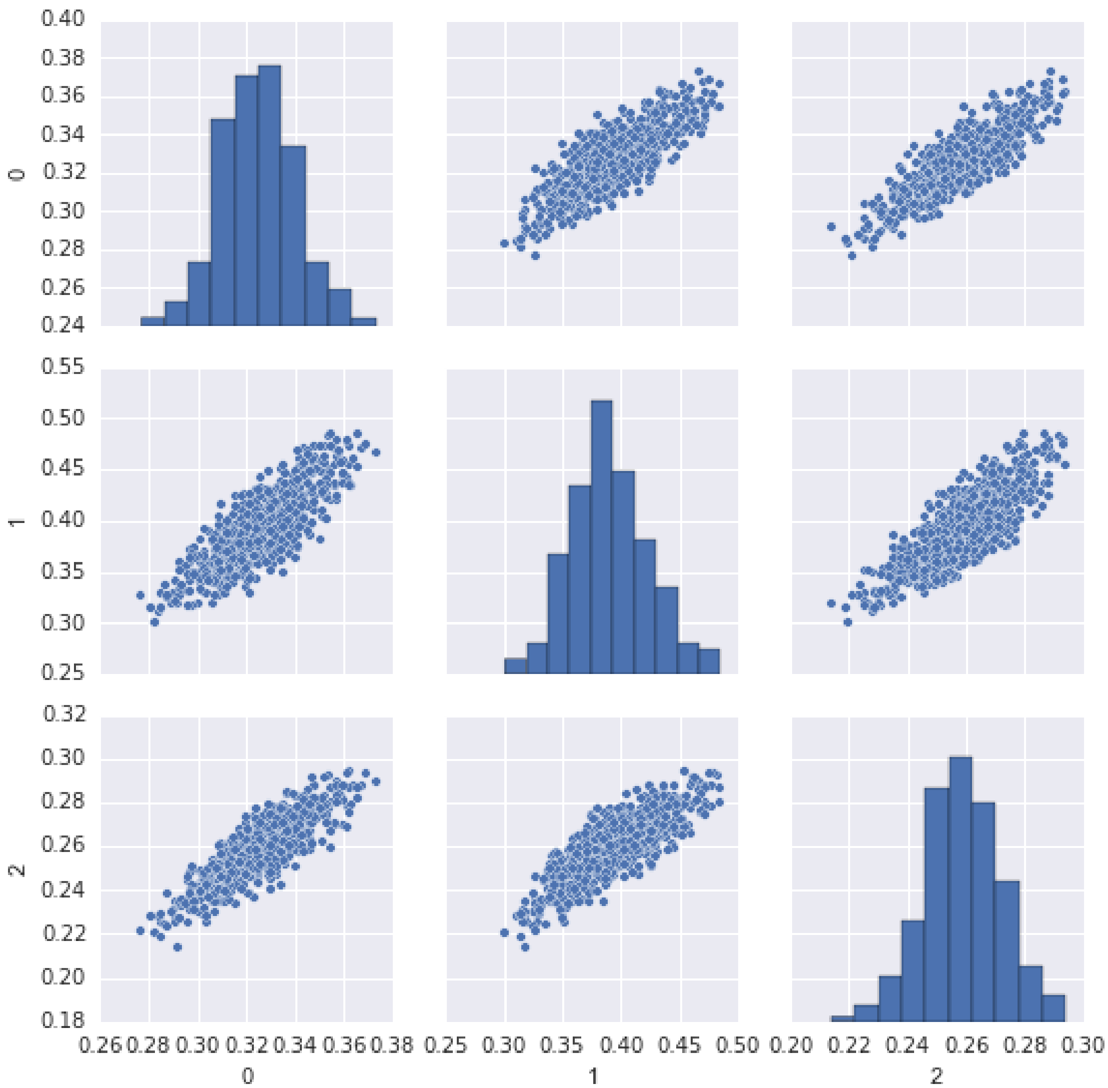
5.1.1 We find that we have a NEGATIVE coefficient for Batting Average.

(This would mean that as your Batting Average goes up, the Runs Scored goes down!)
We assume this has to do with Multicollinearity and, as such, we investigate our inkling...

5.2 We can further investigate this by looking at a "scatterplot matrix" of our predictor variables:

```
In [38]: import seaborn as sns
         sns.set()

         df = pd.DataFrame(X)
         sns.pairplot(df)
```

```
Out[38]: <seaborn.axisgrid.PairGrid at 0x10af21d10>
```



5.3 And we can create a new regression model without this variable (Batting Average):

```
In [22]: X = np.vstack((np.array(moneyball["OBP"]),
                        np.array(moneyball["SLG"]))).T
         rs_model = linear_model.LinearRegression()
         rs_model.fit(X, y)
         print("R-squared[RS_2]: %.4f" % (rs_model.score(X, y)))
         print(rs_model.intercept_, rs_model.coef_)
```

```
R-squared[RS_2]: 0.9296
(-804.62706106223936, array([ 2737.76802227,  1584.90860546]))
```

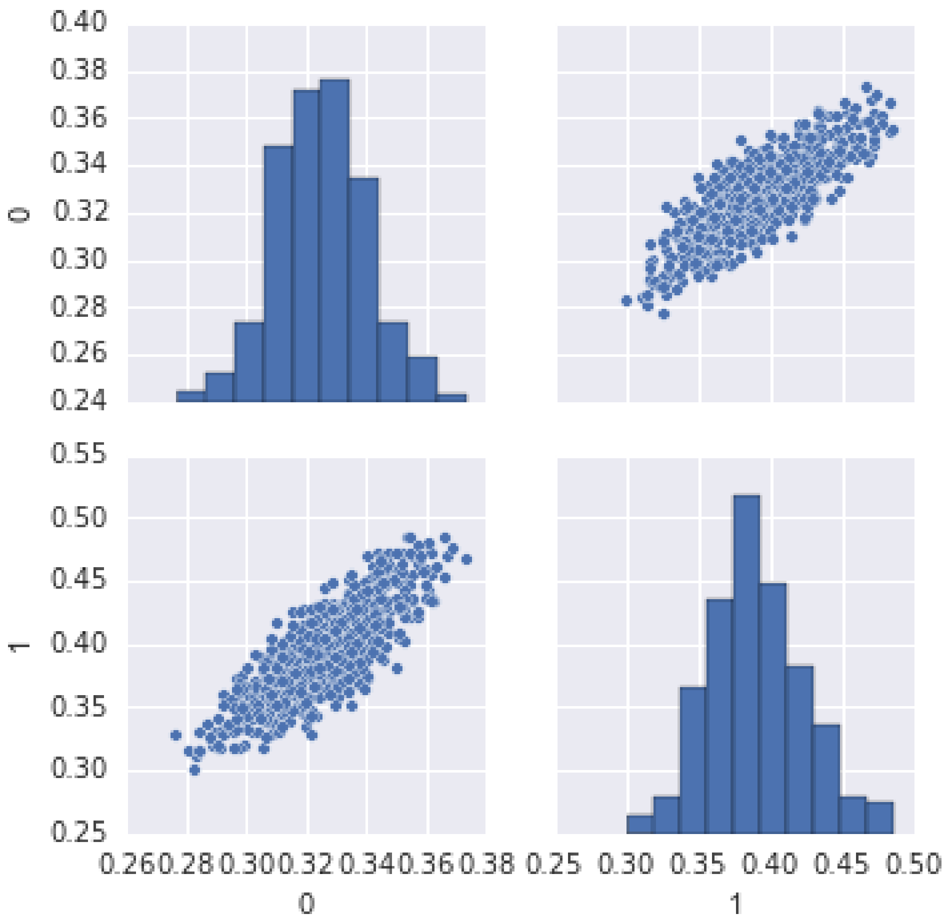
5.4 Our fit is just as good ($R^2 = 0.929$ instead of $R^2 = 0.9302$) and we no longer see the issues with Multicollinearity: the coefficients make sense

5.4.1 We can also take a further look at the two variables that we *are* using to build our model:

On Base Percentage
  -and-
  Slugging Percentage

```
In [45]: df2 = pd.DataFrame(X)
         sns.pairplot(df2)
```

```
Out[45]: <seaborn.axisgrid.PairGrid at 0x10e9ff250>
```



6 Now we can build a model to predict our other main variable:

7 Runs Against

7.0.2 To build thi model, we will -analagously- use the following variables:

-Opponents On Base Percentage (OOBP)

-Opponents Slugging Percentage (OSP)

```
In [24]: moneyball_ra = pd.DataFrame({"OOBP": moneyball["OOBP"],
                                      "OSLG": moneyball["OSLG"],
                                      "RA": moneyball["RA"]})
         moneyball_ra = moneyball_ra.dropna(axis=0)
         y = np.array(moneyball_ra["RA"])
         X = np.vstack((np.array(moneyball_ra["OOBP"]),
                        np.array(moneyball_ra["OSLG"]))).T
         ra_model = linear_model.LinearRegression()
         ra_model.fit(X, y)
         print("R-squared[RA]: %.4f" % (ra_model.score(X, y)))
         print"\n"
         print("Intercept: %.4f" %rs_model.intercept_)
         print("Opponents On Base Percentage Coefficient: %.2f" % ra_model.coef_[0])
         print("Opponents Slugging Percentage Coefficient: %.2f" % ra_model.coef_[1])
```

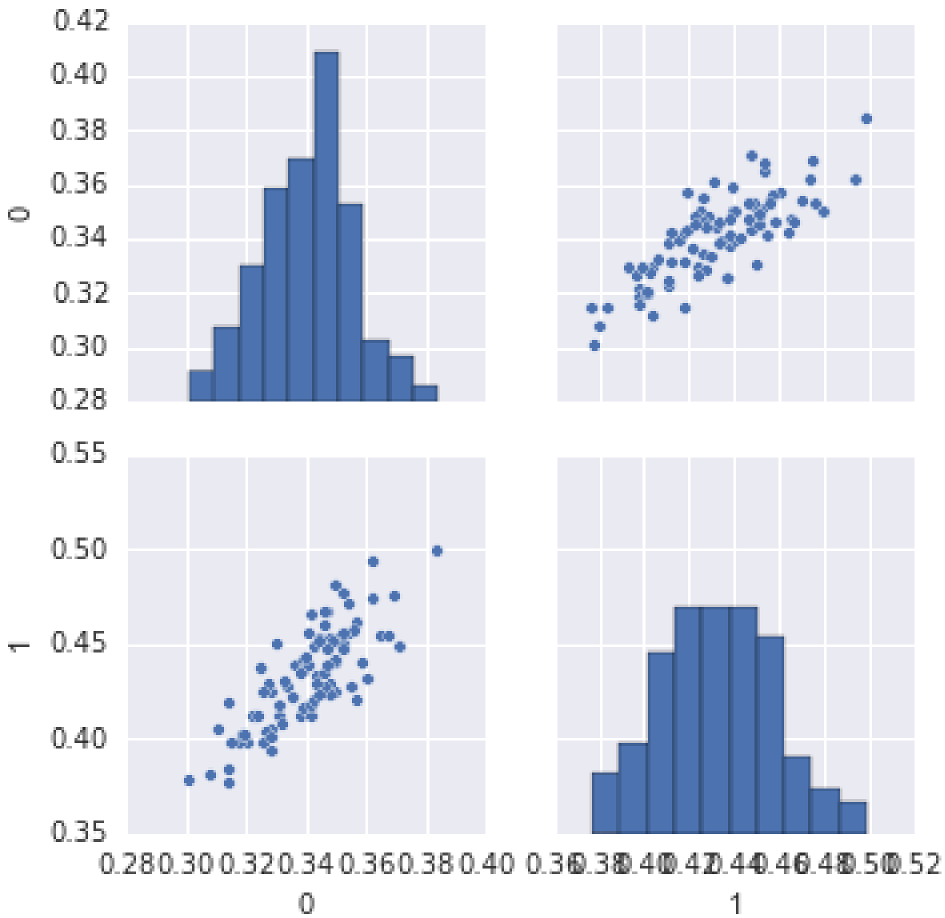
```
R-squared[RA]: 0.9073


Intercept: -804.6271
Opponents On Base Percentage Coefficient: 2913.60
Opponents Slugging Percentage Coefficient: 1514.29
```

```
In [ ]:
```

7.1 And we can look at these variables as well, to understand them better:

```
In [47]: df3 = pd.DataFrame(X)
         sns.pairplot(df3)
```

```
Out[47]: <seaborn.axisgrid.PairGrid at 0x10efba4d0>
```



8 Putting it all together

8.1 Let's take a step back and look at what we've actually done:

8.1.1 -We wanted to be able to see if we could use the past data from Major League Baseball to be able to statistically infer what types of outcomes were needed to produce a playoff entry for a team. To be able to do this, we first found the variable that was most highly correlated (read: predictive) of a playoff entry. Next, we needed to break apart *that* variable into various components (read: other variables) which were highly predictive of it. This allowed us to start breaking apart *what exactly* makes for a playoff-worthy team.

8.1.2 So... we used other variables which were in our BaseballReference.com dataset to try and find which had relationships between one another, and we tried to "discover" or "define" those relationships by building a model (here, a linear regression model) which explained as much of the variance between them as possible.

9 How good do our models do?

Let's predict our Runs Scored using our On Base Percentage: 0.339 and our Slugging Percentage: 0.430.

Let's predict our Runs Against using the Opponents On Base Percentage: 0.307 and our Opponents Slugging Percentage: 0.373.

(These are our 2001 values... so, our best guess going into 2002.)

We'll then take these predicted values to create our predicted Runs Difference value.

Finally, with our predicted Runs Difference, we can make a prediction as to how many Wins we will have!

In [23]:
```
pred_rs = rs_model.predict(np.matrix([0.339, 0.430]))
pred_ra = ra_model.predict(np.matrix([0.307, 0.373]))
pred_rd = pred_rs - pred_ra
pred_wins = reg.predict(np.matrix([pred_rd]))
print("Predicted Runs Scored in 2002: %.2f" % (pred_rs))
print("Predicted Runs Allowed in 2002: %.2f" % (pred_ra))
print("Predicted Wins in 2002: %.2f" % (pred_wins))
```

```
Predicted Runs Scored in 2002: 804.99
Predicted Runs Allowed in 2002: 621.93
Predicted Wins in 2002: 100.24
```

10 To get an idea of how well we fared compared to DePodesta... let's look at our predictions compared to his:

10.1 DePodesta's Estimates:

10.2 Runs Scored: 800-820 (The A's actually scored 800 in 2002.)

10.3 Runs Allowed: 650-670 (The A's actually allowed 653 in 2002.)

10.4 Wins: 93-97 (The A's actually won 103 games.)