int15proj_FINAL_VERSION

July 8, 2019

1 INT15 FINAL PROJECT

1.1 Due Date: June 12, midnight

Factors in Determination of Player Salaries and Win Percentages
Names: Matthew Coleman and Stephen Lantin

1.1.1 We are going to be analyzing the baseball dataset to see if we can determine batting factors which determine players salaries, as well as look at team wins and whether the ratio of runs to earned runs can be used to predit win probabilities of a team

Below are the packages used in our analysis

```
In [1]: import pandas as pd
        import numpy as np

        %matplotlib inline
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.preprocessing import StandardScaler
        from sklearn.decomposition import PCA
        from sklearn.linear_model import RidgeCV, LassoCV, Ridge, Lasso
```

```
In [2]: from sklearn.datasets import load_diabetes
        from sklearn import linear_model
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_squared_error as mse

        from scipy import stats
        import statsmodels.api as sm
```

Reading of CSV's into dataFrames which we are going to explore and then use

```
In [3]: batting = pd.read_csv('Batting.csv')
        #batting[batting['yearID'] == 2018]
        fielding = pd.read_csv('FieldingOF.csv')
        playerinfo = pd.read_csv('People.csv')
        hof_df = pd.read_csv('HallOfFame.csv')
```

```
In [4]: batting = pd.read_csv('Batting.csv')
        #batting[batting['yearID'] == 2018]
        fieldingof = pd.read_csv('FieldingOF.csv')
        fieldingif = pd.read_csv('FieldingOF.csv')
        playerinfo = pd.read_csv('People.csv')
        #playerinfo.head()
        pitching = pd.read_csv('Pitching.csv')
        #pitching.tail(100)
        salaries = pd.read_csv('Salaries.csv')
        #salaries.head(100)
        salary = salaries['salary']
        #batting.head()
        #batting['salary'] = salary
        teams = pd.read_csv('Teams.csv')
```

Here we merge the salaries from the salary table to the batting table on player ID and year ID so we can analyze the different statistics against salary

```
In [5]: final_salaries = batting.merge(salaries, how = 'left', on = ['playerID','yearID'], suf

        final_salaries.drop(list(final_salaries.filter(regex='_y$')), axis=1, inplace=True)
        #final_salaries = final_salaries[final_salaries['salary_x'].notnull()]
        #final_salaries['yearID'].unique()
        final_salaries = final_salaries[final_salaries['salary'].notnull()]
```

Here, we choose the salaries that are in the year 1990. We chose to do this because there was too much over plotting if we chose more than one year. We proceded to fill all the NaN values with 0. We chose this under the assumption that if there was a blank spot, it was because they did not have any stats under there.

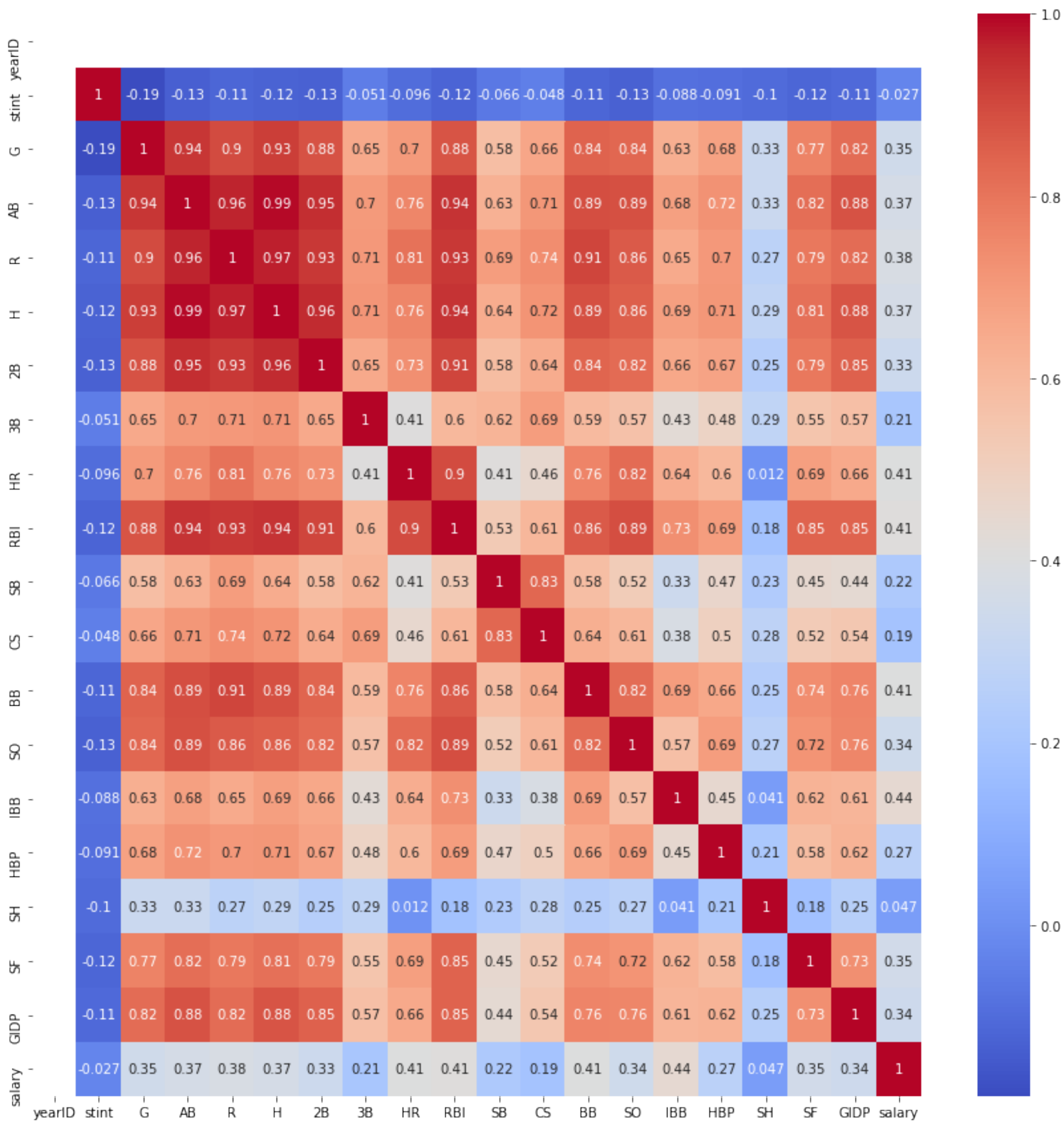
```
In [6]: #final_salaries
        #salaries_95 = final_salaries[(final_salaries['yearID'] >= 1990) & (final_salaries['ye
        #salaries_00 = final_salaries[(final_salaries['yearID'] >= 1995) & (final_salaries['ye
         # & (final_salaries['yearID'] < 1995)]
        #salaries_40 = final_salaries[(final_salaries['yearID'] >= 1940) & (final_salaries['ye

        salaries_90 = final_salaries[(final_salaries['yearID'] == 1990)]
        salaries_90 = salaries_90.fillna(0)

        #too much over plotting, So im just going to look at the data from the 1990s. Maybe we
        # 90-95, or something along those lines. Christian said its going to be hard to get an
        #the years as a result of oveplotting.

        #NOTE:If we decide that we need to look at data across multiple years, we should look
        #want to see how other players/teams perform in relationship to other teams, we should
        #that we can see who is the furthest team/player from the average

In [7]: #More Exploratory Code, did not use any of this in the final analysis
```

```
#standardized_salary = (salaries_95['salary'] -
#                        (np.sum(salaries_95['salary'])/len(salaries_95))) #/np.sqrt(le

#standardized_salary
#here I am going to standardize the salary column
#(salaries_95['salary'] -np.sum(salaries_95['salary'])/len(salaries_95))

#normalize variables that depend on the number of players/team (note made by christian

#this code below is irrelevant, I want to keep it as reference but I do not plan on us
#std_values = (salaries_95['salary'] -(np.sum(salaries_95['salary'])/len(salaries_95))

#salaries_95['standardized'] = std_values
#salaries_95
```

The code below was to create a heatmap based on the correlation between variables. Because we were planning to use linear regression, this was a useful diagnostic plot for determining which variables to explore

In [8]:
```
#Here, we look at the correlations between all the variables that will be relevant to
#If there is anything that catches out eye, we will delve into possible relationships

bat_corr = salaries_90.corr()
fig, ax = plt.subplots(figsize = (15,15))
sns.heatmap(bat_corr, annot = True,cmap = 'coolwarm')
```

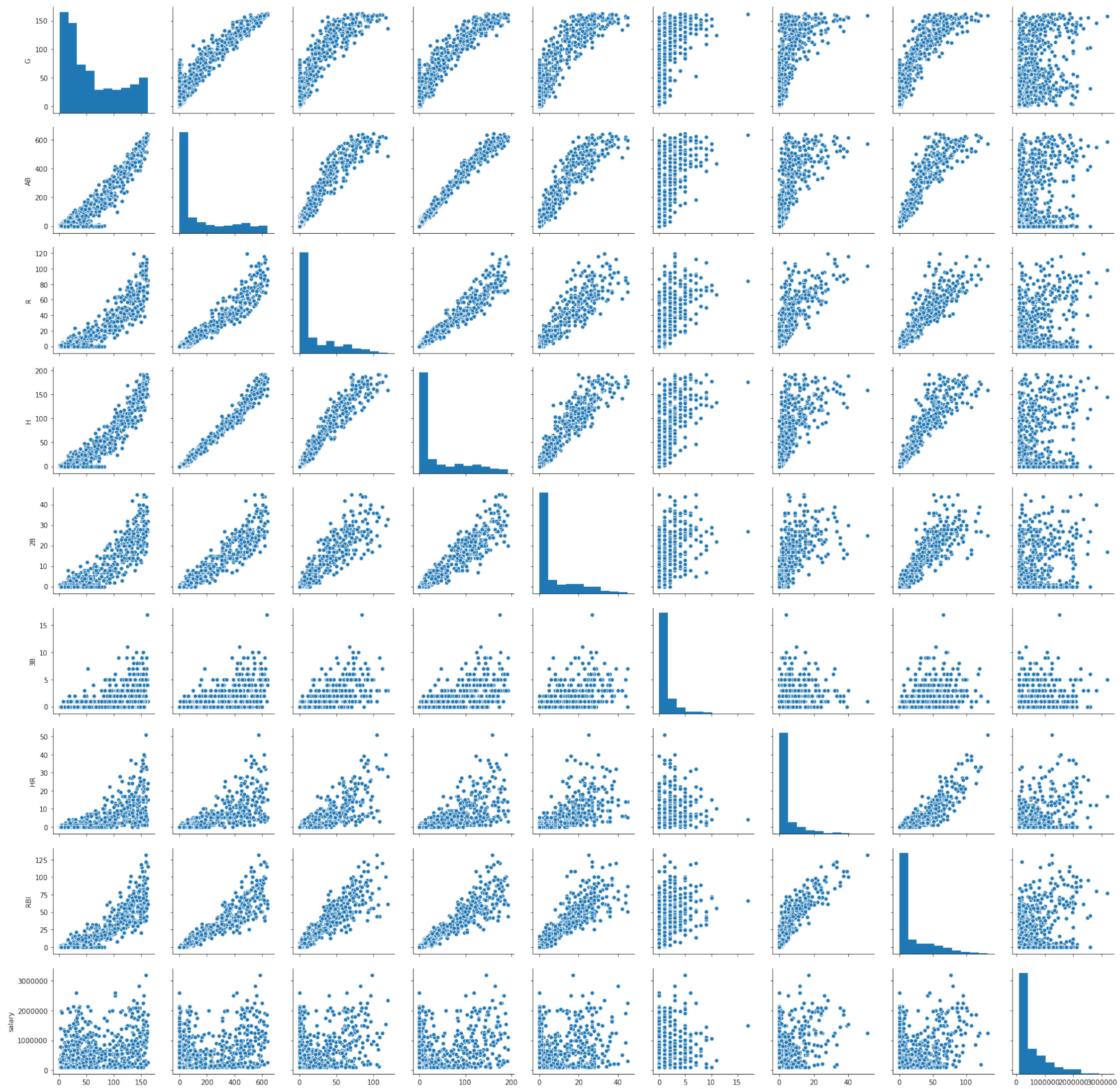
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff846a019b0>



```
In [9]: #desired_columns = ['G','AB', 'R', 'H', '2B', '3B', 'HR', 'RBI', 'SB', 'SB', 'CS', 'BB

        hitting_stats = ['G','AB', 'R', 'H', '2B', '3B', 'HR', 'RBI', 'salary'] #, 'standardiz
        other_stats = ['SB', 'SB', 'CS', 'BB', 'SO', 'salary', 'standardized']
        batting_info = salaries_90.loc[:, hitting_stats]
```

Pairplot of the hitting data, visible issues of overplotting in some aspects. In stast AB through RBI compared against salary, there are clear issues with overplotting, and a line of 0 values for all of the stats. This is where we discovered that the line of zeroes was most likely pitchers, and decided to remove these data points

```
In [10]: sns.pairplot(salaries_90.loc[:, hitting_stats])
         plt.show()
```



As mentioned above, looking at our analyses of the data, we noticed a “wall” of 0 values for almost variables plotted against salary. It was determined that there are many players who are not good at batting but still get paid more because of other measures of value. One of these players are pitcher, so we removed them from our analyses to look at players who could be evaluated more on their offensive aspects than their defensive ones.

```
In [11]: nonpitch_90 = salaries_90[~salaries_90['playerID'].isin(pitching['playerID'])]
         #len(sal_90_nonpitch)
         #the point of this cell is to take out all the players who are in the pitching datafr
         #variables here is pitchers salary, where they may not have as many batting statistic
         #highly because their pitching is valuable
         test_hit = nonpitch_90
         sample_50 = nonpitch_90.sort_values(by ='HR',ascending = False).iloc[0:50,:]
         sample_50.shape #[['playerID','HR']]
```

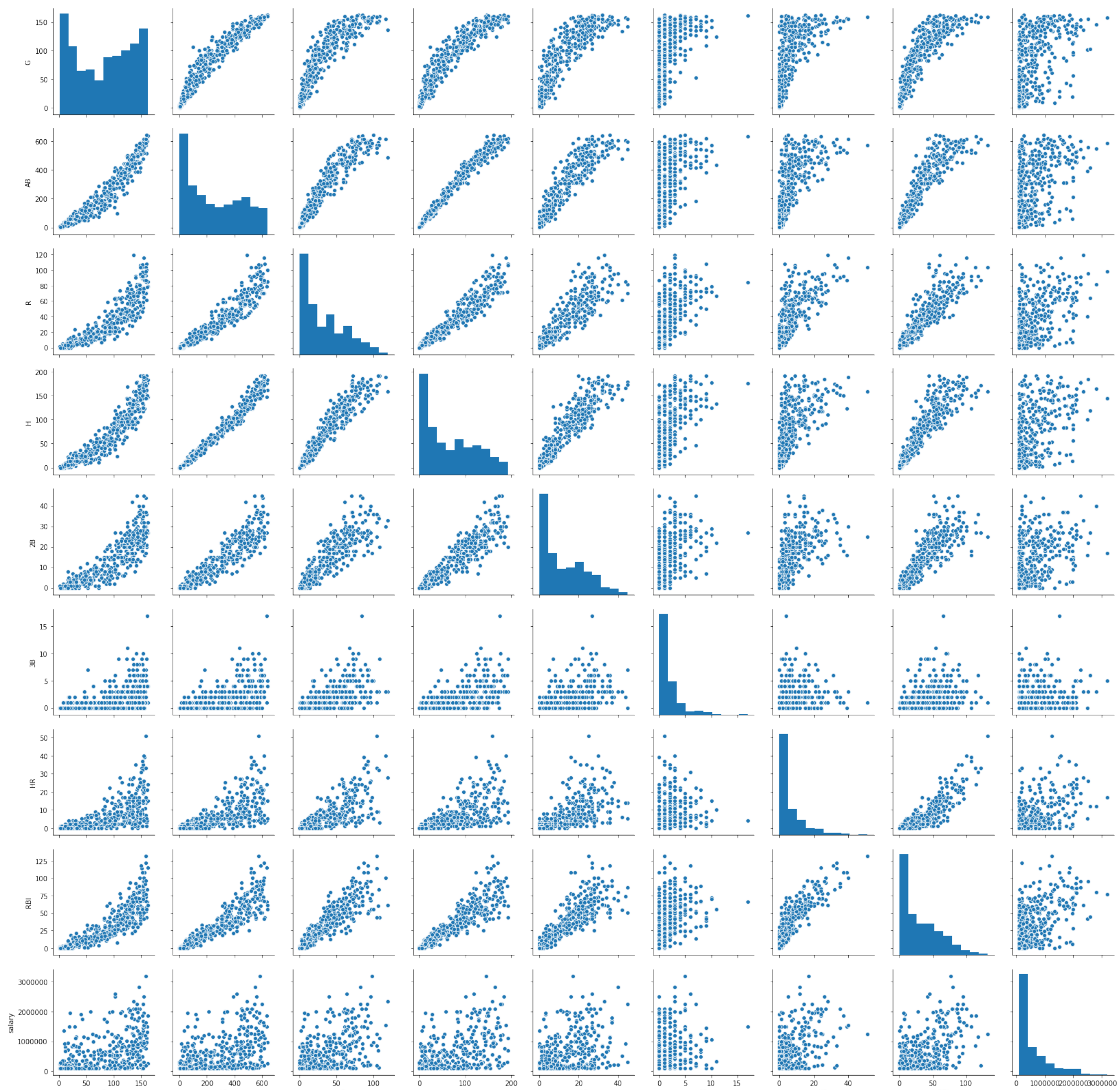
Out[11]: (50, 23)

1.1.2 Here, we are splitting our data into data_tr, which will be our training data, and then data_te, which will be our tesitng data. By splitting the data this way, we will be able to train the model and then use the test data to test the model.

```
In [12]: data_tr, data_te = train_test_split(nonpitch_90, train_size = .75, test_size = .25)
```

Creating a pairplot of the batting data without pitchers included shows that there is no longer an issue with the vertical line of zero values. Many of the plots have correlation, even if very weak.

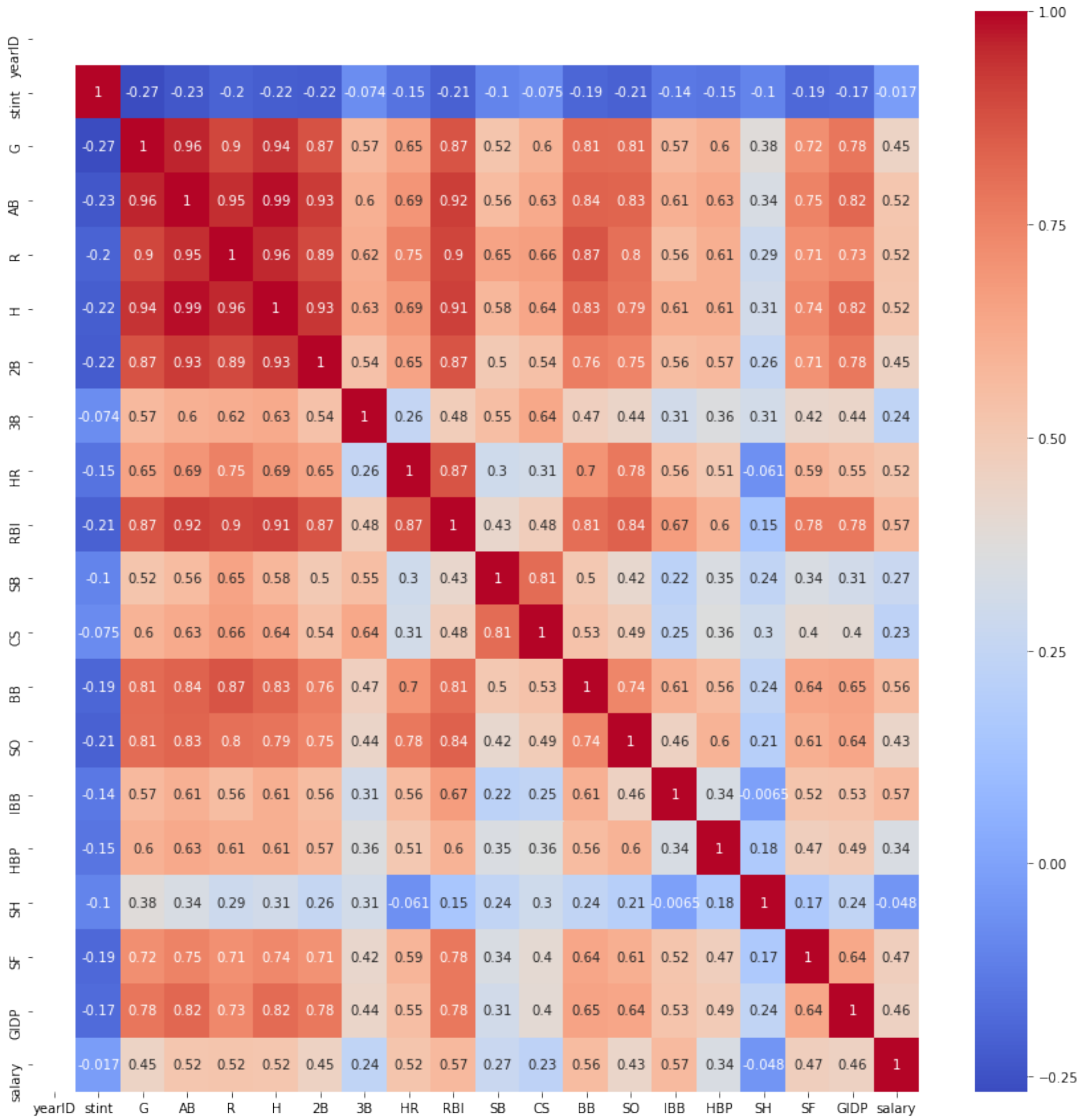
```
In [13]: # We can use this pairplot to look at the relationships between vairables

         sns.pairplot(nonpitch_90.loc[:, hitting_stats])
         plt.show()
         #this pairplot is not perfect, but it is by far an improvement from the one before
```



Correlation matrix of the nonpitch data. Many of the correlations are better.

```
In [14]: nonpitchbat_corr = nonpitch_90.corr()
         fig, ax = plt.subplots(figsize = (15,15))
         sns.heatmap(nonpitchbat_corr, annot = True, cmap = 'coolwarm')
```

```
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff841612240>
```



We fit all the models that we are interested in below.

```
In [15]: fullmodel = sm.OLS(data_tr['salary'], data_tr[['AB','R','H','2B','3B','HR','RBI','SB'
                                                         'CS','BB','SO','IBB','HBP','SH
```

```
hit_model = sm.OLS(data_tr['salary'], data_tr[['2B','3B','HR']]).fit()

nonhit = sm.OLS(data_tr['salary'], data_tr[['R','H','RBI','SB','CS','BB','IBB']]).fit

ibb_model = sm.OLS(data_tr['salary'], data_tr['IBB']).fit()

rbi_model = sm.OLS(data_tr['salary'], data_tr['RBI']).fit()


newfull = sm.OLS(data_tr['salary'], data_tr[['AB','2B','SB',
                                              'CS','BB','IBB','SH']]).fit()

#fullmodel.summary()
#hit_model.summary()
```

```
In [16]: fullmod_predictions = fullmodel.predict(data_te[['AB','R','H','2B','3B','HR','RBI','SI
                                                  'CS','BB','SO','IBB','HBP','SH
         hit_predictions = hit_model.predict(data_te[['2B','3B','HR']])
         nonhit_predictions = nonhit.predict(data_te[['R','H','RBI','SB','CS','BB','IBB']])
         ibb_predictions = ibb_model.predict(data_te['IBB'])
         rbi_predictions = rbi_model.predict(data_te['RBI'])
         test_salaries = data_te['salary']
         #fig, ax = plt.subplots(figsize = (10,10))
         #sns.scatterplot(x=predictions.index, y=predictions, color = 'r')
         #sns.scatterplot(x=predictions.index, y=fullmod_predictions)
```

Creating a user defined function below for giving the percentage of underestimates and over estimates

```
In [17]: ##defined function to find the percent above and percent below statistics

         def overlower_estims(predictions,actual):
             difference = predictions - actual
             predict_error = []

             for value in difference:
                 if value >= 0:
                     predict_error.append(1)
                 elif value < 0:
                     predict_error.append(0)

             percent_below = (len(predict_error)-np.sum(predict_error))/len(predict_error)
             print("Percent of Under Estimates: ", 100*percent_below, ", Percent of Over Estima
```

Creating a user defined function to test the accuracy of our model at a $100,000 tolerance level.

```
In [18]: def percent_correct(predictions, actual):
             correct_ibb = np.isclose(predictions, actual, atol = 100000)
             print("Percent of correct predictions: ", 100*(np.sum(correct_ibb)/len(correct_ib
```

Precent of under estimates and correct predictions for the full model

```
In [19]: #FULL MODEL PREDICTIONS
         overlower_estims(fullmod_predictions, test_salaries)
         percent_correct(fullmod_predictions, test_salaries)
         rmsef= np.sqrt(mse(test_salaries, fullmod_predictions))
         rmsef
```

```
Percent of Under Estimates:  68.10344827586206 , Percent of Over Estimates:  31.89655172413793
Percent of correct predictions:  36.206896551724135
```

```
Out[19]: 389436.0183545164
```

Precent of under estimates and correct predictions for the hit model

```
In [20]: #HIT PREDICTIONS
         overlower_estims(hit_predictions, test_salaries)
         percent_correct(hit_predictions, test_salaries)
         rmsehit= np.sqrt(mse(test_salaries, hit_predictions))
         rmsehit
```

```
Percent of Under Estimates:  72.41379310344827 , Percent of Over Estimates:  27.5862068965517
Percent of correct predictions:  37.06896551724138
```

```
Out[20]: 481656.7625021634
```

Precent of under estimates and correct predictions for the non full model

```
In [21]: #THIS IS THE PERCENT BELOW FOR THE NON-FULL MODEL WITH NON HITTING STATS SUCH AS SB, 
         overlower_estims(nonhit_predictions, test_salaries,)
         percent_correct(nonhit_predictions, test_salaries)
         rmsenf = np.sqrt(mse(test_salaries, nonhit_predictions))
         rmsenf
```

```
Percent of Under Estimates:  72.41379310344827 , Percent of Over Estimates:  27.5862068965517
Percent of correct predictions:  29.310344827586203
```

```
Out[21]: 410159.2936268346
```

Precent of under estimates and correct predictions for the intentional ball model

```
In [22]: #INTENTIONAL PREDICTIONS
         overlower_estims(ibb_predictions, test_salaries)
         percent_correct(ibb_predictions, test_salaries)
         rmseibb = np.sqrt(mse(test_salaries, ibb_predictions))
         rmseibb
```

```
Percent of Under Estimates:  80.17241379310344 , Percent of Over Estimates:  19.82758620689655
Percent of correct predictions:  31.896551724137932
```

```
Out[22]: 495166.50024517084
```

Percent of under estimates and correct predictions for the runs batted in model

```
In [23]: overlower_estims(rbi_predictions, test_salaries)
         percent_correct(rbi_predictions, test_salaries)
```

```
Percent of Under Estimates:  70.6896551724138 , Percent of Over Estimates:  29.31034482758621
Percent of correct predictions:  36.206896551724135
```

1.2 Note: We attempted running the same analyses that we did on the batting data, except with the pitching data. We encountered many of the same problems, and overall could not find anything interesting related to what we set out to explore. As a result, we did not include any analysis relating to pitching in our report. Here is the code, showing there was exploration into pitching.

```
In [24]: pitching_salaries = pitching.merge(salaries, how = 'left', on = ['playerID','yearID']
         pitching_salaries.drop(list(pitching_salaries.filter(regex='_y$')), axis=1, inplace=T
```

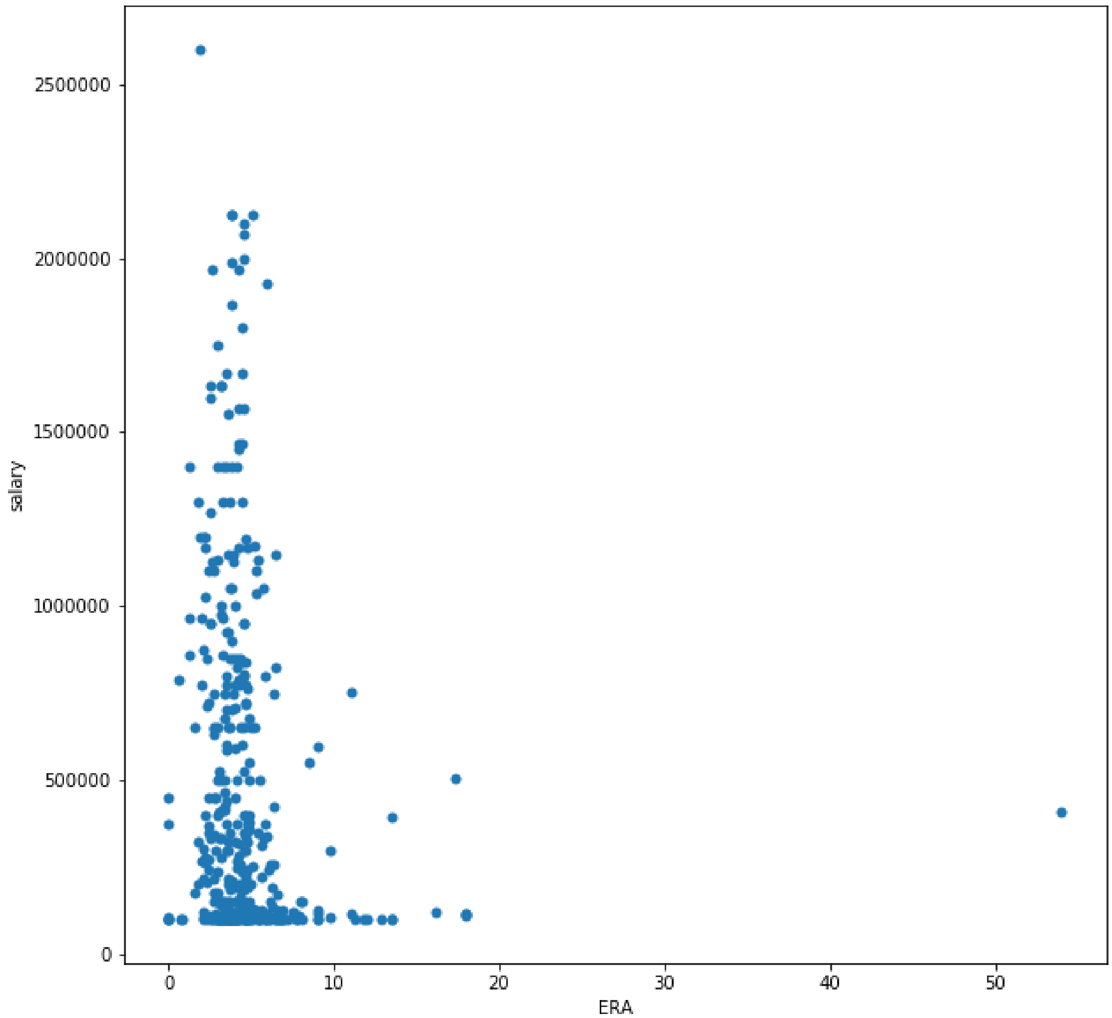
```
In [25]: pitching_90 = pitching_salaries[pitching_salaries['yearID'] == 1990]
         pitching_90 = pitching_90[pitching_90['salary'].notnull()]

         pitching_90.head(10)
         pitching_90.columns
```

```
Out[25]: Index(['playerID', 'yearID', 'stint', 'teamID', 'lgID', 'W', 'L', 'G', 'GS',
                'CG', 'SHO', 'SV', 'IPouts', 'H', 'ER', 'HR', 'BB', 'SO', 'BAOpp',
                'ERA', 'IBB', 'WP', 'HBP', 'BK', 'BFP', 'GF', 'R', 'SH', 'SF', 'GIDP',
                'salary'],
               dtype='object')
```

```
In [26]: pitching_90 = pitching_salaries[pitching_salaries['yearID'] == 1990]
         pitch_tr, pitch_te = train_test_split(pitching_90, train_size = .75, test_size = .25)
         #after doing some research, it seems that the 50 ERA point is highly leveraged point
         #as I could not find anything relating to a player 'Dave Martinez having a 50 ERA'
         #remove the outlier
```

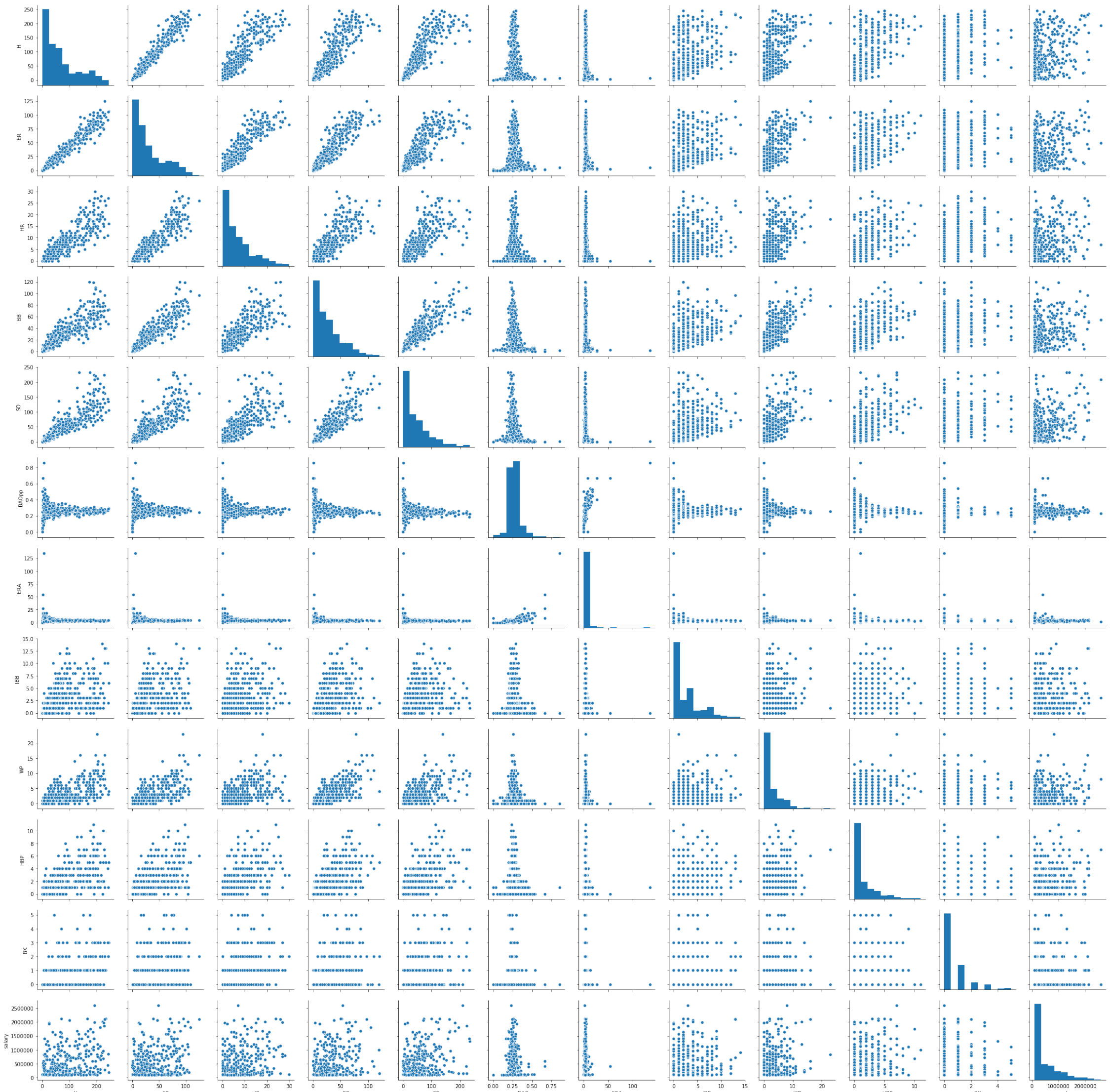
```
In [27]: fig, ax = plt.subplots(figsize = (10,10))
         sns.scatterplot(x=pitching_90['ERA'] , y=pitching_90['salary'],
                         sizes=(1, 8), linewidth=0,
                          ax=ax)
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff8477bcef0>
```

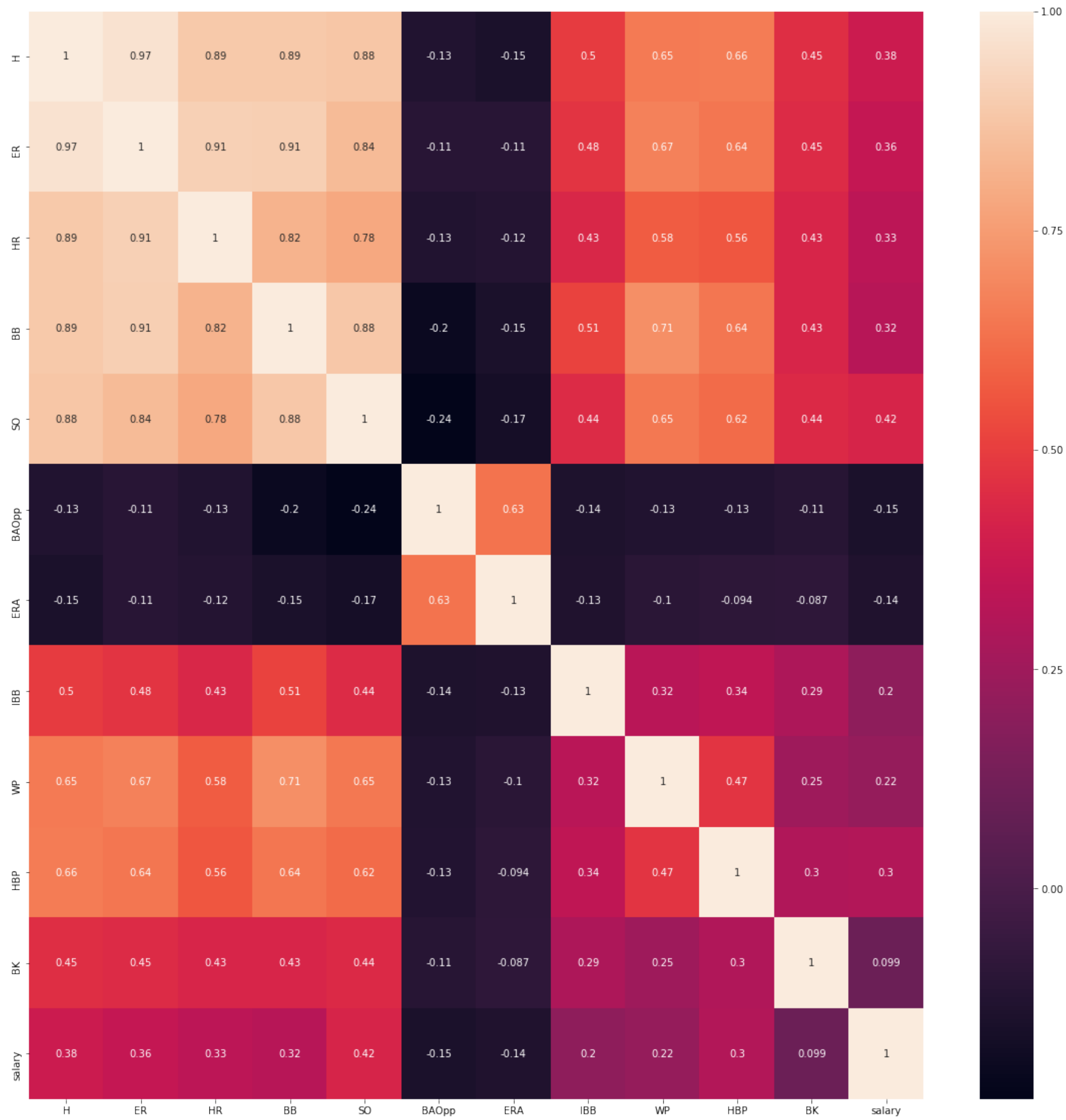



```
In [28]: sns.pairplot(pitching_90.loc[:,['H', 'ER', 'HR', 'BB', 'SO', 'BAOpp',
                'ERA', 'IBB', 'WP', 'HBP', 'BK','salary']])
         plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/numpy/lib/histograms.py:824: RuntimeWarning: invalid va
  keep = (tmp_a >= first_edge)
/opt/conda/lib/python3.6/site-packages/numpy/lib/histograms.py:825: RuntimeWarning: invalid va
  keep &= (tmp_a <= last_edge)
```



```
In [29]: pit_corr = pitching_90[ ['H', 'ER', 'HR', 'BB', 'SO', 'BAOpp',
                 'ERA', 'IBB', 'WP', 'HBP', 'BK','salary']].corr()
         fig, ax = plt.subplots(figsize = (20,20))
         sns.heatmap(pit_corr, annot = True)

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff835769748>
```



1.2.1 End of pitching analysis

1.3 Working with team data

Choosing the team data from 1990 to 2000 and adding a fracwin column, this column is calculated as $\frac{Wins}{Wins+Losses}$

```
In [30]: teams_full = teams[(teams['yearID']>=1990) & (teams['yearID']<2000)]
         teams_90 = teams_full.loc[:,'R':'FP']
         teams_90['W'] = teams['W']
         teams_90['L'] = teams['L']
```

```
teams_90['fracwin'] = (teams['W']/(teams['W'] + teams['L'])) # (wins/(wins + losses))
#teams_90['fracwin']
```

In [31]:
```
#function to add a 1 if more wins than losses.
def wins(row):
    if row['fracwin'] >= .5:
        val = 1
    elif row['fracwin'] < .5:
        val = 0
    return val
```

In [32]:
```
#adding a row of dummy variables to whether they won most of their games or not
teams_90['target'] = teams_90.apply(wins, axis = 1)
teams_90.head()
```

Out[32]:
```
         R    AB     H   2B  3B   HR     BB      SO     SB    CS  ...  HRA  \
2047   682  5504  1376  263  26  162  473.0  1010.0   92.0  55.0  ...  128
2048   669  5410  1328  234  22  132  660.0   962.0   94.0  52.0  ...  161
2049   699  5516  1502  298  31  106  598.0   795.0   53.0  52.0  ...   92
2050   690  5570  1448  237  27  147  566.0  1000.0   69.0  43.0  ...  106
2051   682  5402  1393  251  44  106  478.0   903.0  140.0  90.0  ...  106

       BBA  SOA    E   DP     FP   W   L   fracwin  target
2047   579  938  158  133  0.974  65  97  0.401235       0
2048   537  776   93  151  0.985  76  85  0.472050       0
2049   519  997  123  154  0.980  88  74  0.543210       1
2050   544  944  142  186  0.978  80  82  0.493827       0
2051   548  914  124  169  0.980  94  68  0.580247       1

[5 rows x 30 columns]
```

In [33]:
```
win_tr, win_te = train_test_split(teams_90, train_size = .75, test_size = .25)
win_tr.head()
```

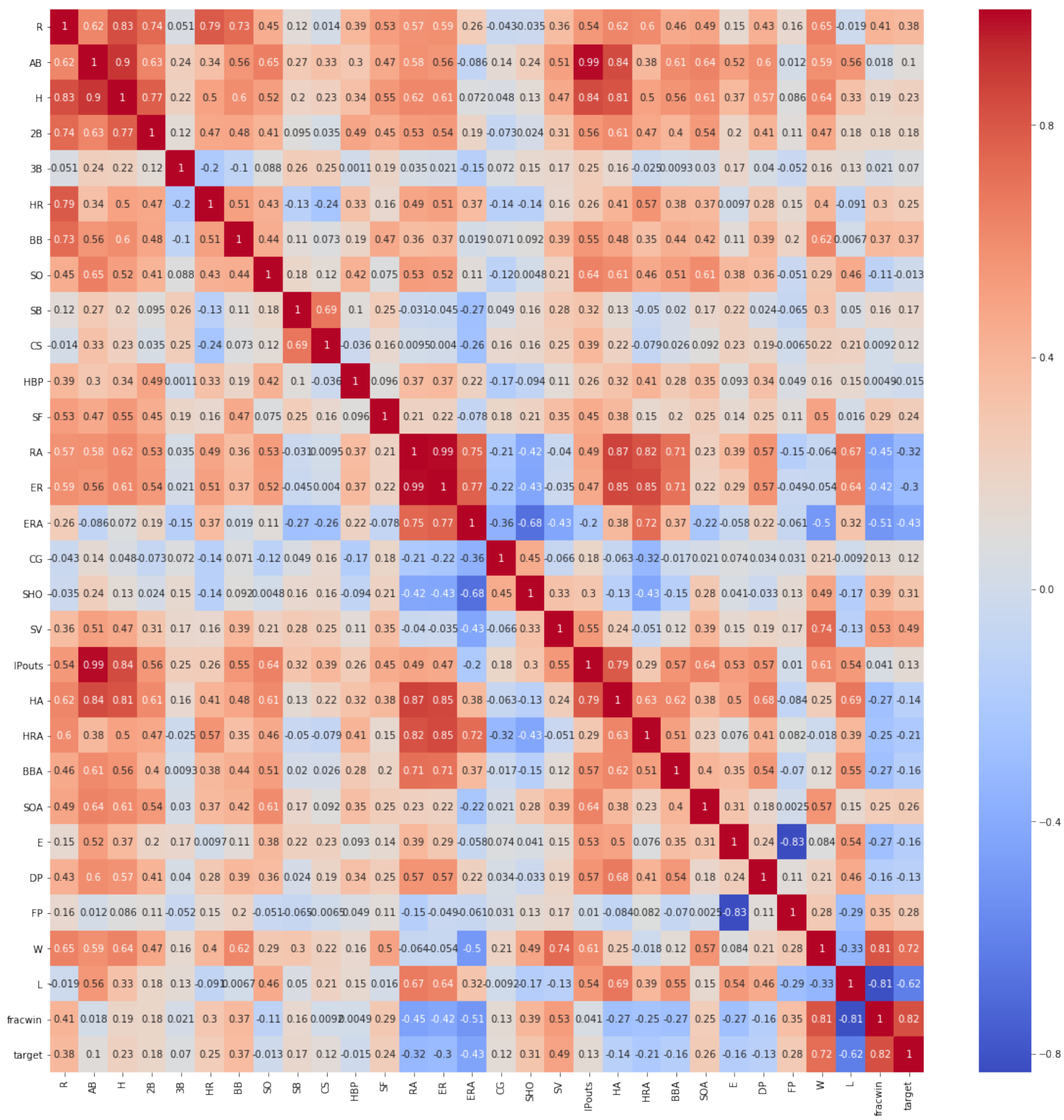
Out[33]:
```
         R    AB     H   2B  3B   HR     BB      SO     SB    CS  ...  HRA  \
2237   829  5628  1531  279  25  161  617.0   953.0  126.0  72.0  ...  202
2238   791  5528  1490  268  37  174  597.0  1160.0  108.0  58.0  ...  111
2243   651  5484  1386  269  27  142  518.0  1113.0  190.0  67.0  ...  173
2068   640  5474  1419  251  26  107  596.0   749.0  105.0  51.0  ...  120
2186   693  4963  1315  267  39  158  440.0   953.0  105.0  37.0  ...  162

       BBA   SOA    E   DP     FP    W   L   fracwin  target
2237   605  1050  123  140  0.980   84  78  0.518519       1
2238   450  1196  114  136  0.982  101  61  0.623457       1
2243   558  1159  106  129  0.982   76  86  0.469136       0
2068   606  1064  130  152  0.979   77  85  0.475309       0
2186   518   926  115  115  0.979   73  71  0.506944       1

[5 rows x 30 columns]
```

Here, we train_test_split the win data.

Below is the correlation plot for the team statistics The target column is going to be the target column for our PCA analysis of wins

In [34]:
```
teamcorr = teams_90.corr()
fig, ax = plt.subplots(figsize = (20,20))
sns.heatmap(teamcorr, annot = True, cmap = 'coolwarm')
```

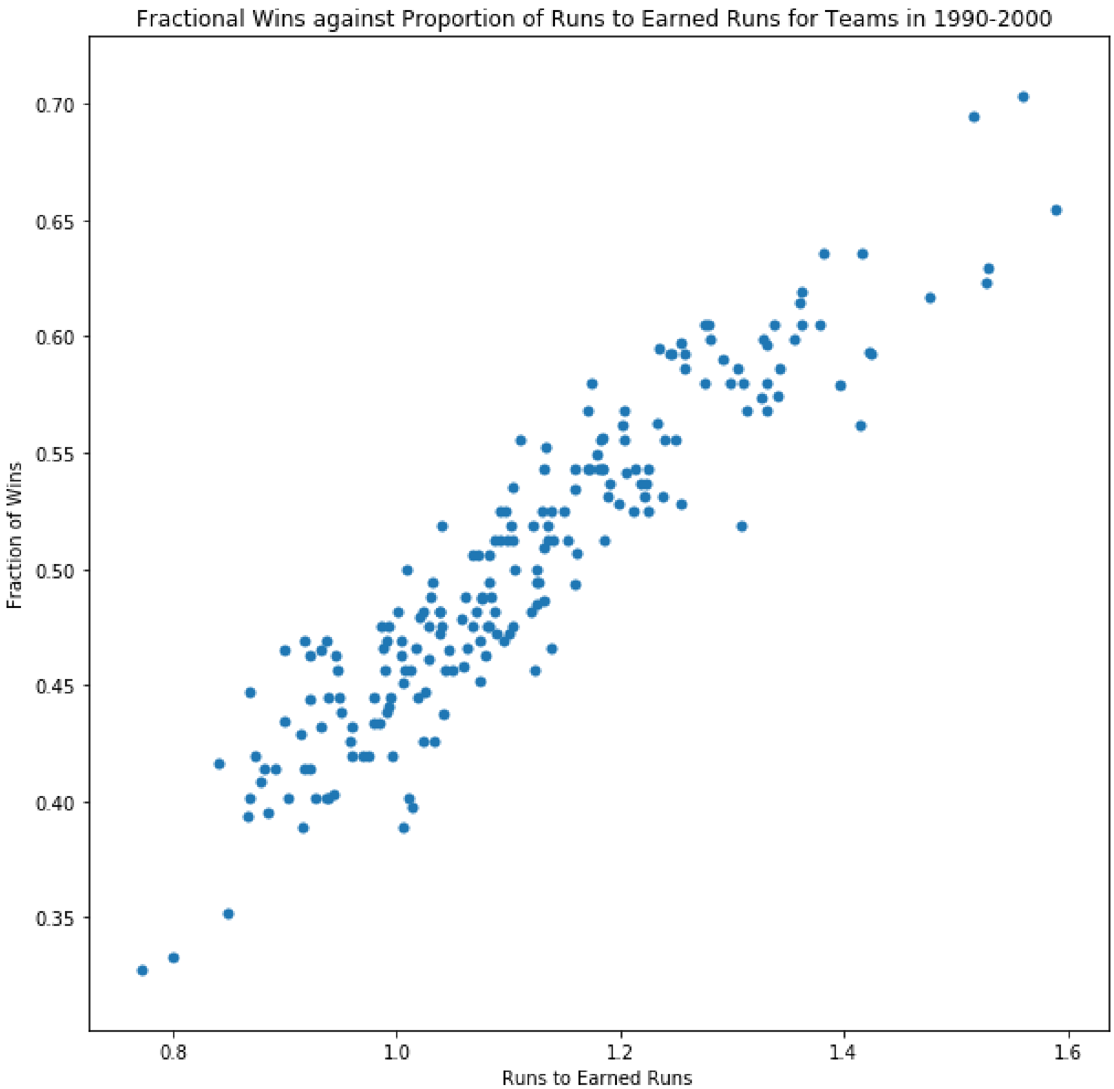
Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff8343d9278>



In [35]:
```
fig, ax = plt.subplots(figsize = (10,10))
sns.scatterplot(x=(win_tr['R']/win_tr['ER']) , y=win_tr['fracwin'],
                    sizes=(1, 8), linewidth=0, ax=ax)
```

```
plt.title("Fractional Wins against Proportion of Runs to Earned Runs for Teams in 1990
plt.ylabel("Fraction of Wins")
plt.xlabel("Runs to Earned Runs")
```

Out[35]: Text(0.5, 0, 'Runs to Earned Runs')



After looking at this plot, lets run a simple linear regression of this data.

In [36]:
```
ratio_model = sm.OLS(win_tr['fracwin'],win_tr['R']/win_tr['ER']).fit()

ratio_model.summary()
```

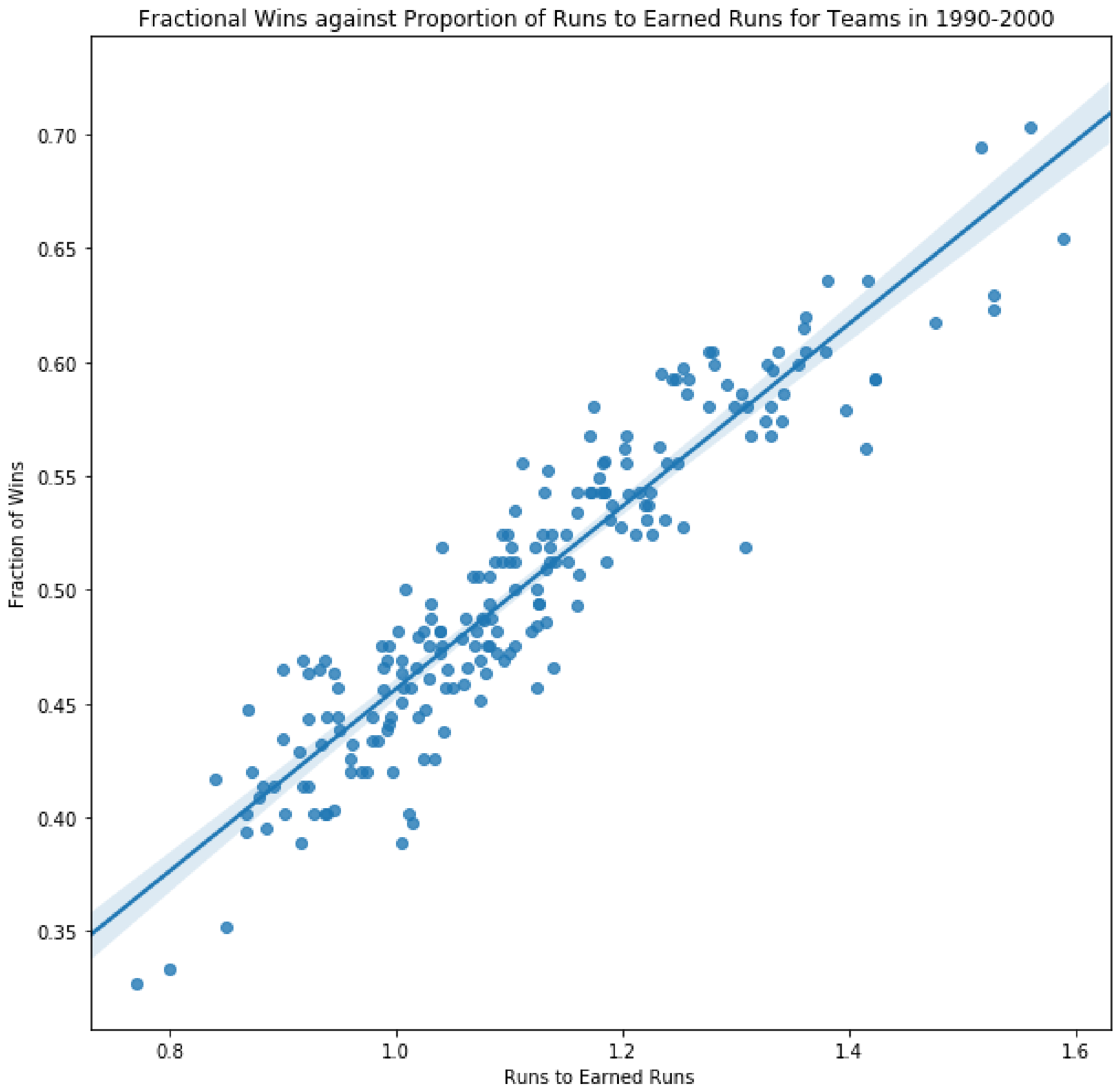
Out[36]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
```

```
==============================================================================
Dep. Variable:                fracwin   R-squared:                       0.997
Model:                            OLS   Adj. R-squared:                  0.997
Method:                 Least Squares   F-statistic:                 7.841e+04
Date:                Tue, 18 Jun 2019   Prob (F-statistic):          5.80e-269
Time:                        18:34:59   Log-Likelihood:                 463.97
No. Observations:                 208   AIC:                            -925.9
Df Residuals:                     207   BIC:                            -922.6
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.4498      0.002    280.011      0.000       0.447       0.453
==============================================================================
Omnibus:                        3.759   Durbin-Watson:                   2.019
Prob(Omnibus):                  0.153   Jarque-Bera (JB):                3.443
Skew:                          -0.308   Prob(JB):                        0.179
Kurtosis:                       3.133   Cond. No.                         1.00
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
"""
```

Now, the plot of the fitted line on the scatterplot

In [37]:
```
fig, ax = plt.subplots(figsize = (10,10))
sns.regplot(x=(win_tr['R']/win_tr['ER']) , y=win_tr['fracwin'])
plt.title("Fractional Wins against Proportion of Runs to Earned Runs for Teams in 1990
plt.ylabel("Fraction of Wins")
plt.xlabel("Runs to Earned Runs")
```

Out[37]:
```
Text(0.5, 0, 'Runs to Earned Runs')
```




```
In [38]: #Fucntion for the percent correct of the win prediction at a 5% tolerance and at a 1%
         def percent_correct_wins_5(predictions, actual):
             correct_ibb = np.isclose(predictions, actual, atol = .05)
             print("Percent of correct predictions at a 5% tolerance level: ", 100*(np.sum(cor

         #Fucntion for the percent correct of the win prediction
         def percent_correct_wins_1(predictions, actual):
             correct_ibb = np.isclose(predictions, actual, atol = .01)
             print("Percent of correct predictions at a 1% tolerance level: ", 100*(np.sum(cor
```

```
In [39]: ratio_pred = ratio_model.predict((win_te['R']/win_te['ER']))
         overlower_estims(ratio_pred, win_te['fracwin'])
         percent_correct_wins_5(ratio_pred, win_te['fracwin'])
         percent_correct_wins_1(ratio_pred, win_te['fracwin'])
```

```
rmsef= np.sqrt(mse(win_te['fracwin'], ratio_pred))
rmsef
```

```
Percent of Under Estimates:  42.857142857142854 , Percent of Over Estimates:  57.14285714285714
Percent of correct predictions at a 5% tolerance level:  87.14285714285714
Percent of correct predictions at a 1% tolerance level:  25.71428571428571
```

```
Out[39]: 0.03187252007689644
```

At a 5% tolerance, the model predicts the correct values about 90% of the time. At a 1% tolerance level, the model predicts the correct value 30 percent of the time. This means that if we want a probability for a team winning within 5% in a game based on the ratio of runs to earned runs, then our accuracy would be about 90 percent. If we wanted to predict the probability of a team winning within a margin of 1%, then we would be correct about 30 percent of the time.

Lets run some PCA on the win data First, lets separate the features from the data set

```
In [40]: features = teams_90.loc[:,'R':'FP'].values
         target = teams_90['target'].values
```

```
In [41]: #standardizing the features
         X = StandardScaler().fit_transform(features)
```

```
In [42]: #Performing the PCA, we will choose 2 components
         pca = PCA(n_components = 10)

         principal_components = pca.fit_transform(X)

         principal_comp_df = pd.DataFrame(data = principal_components, columns = ['PC 1', 'PC 2
                                                                                 'PC 5', 'PC 6
                                                                                 'PC 9', 'PC 1

         principal_comp_df['target'] = target

         pc_df = principal_comp_df

         pc_df.head(10)
```

```
Out[42]:        PC 1      PC 2      PC 3      PC 4      PC 5      PC 6      PC 7  \
         0 -0.241528  0.792956  3.288464 -2.149942  1.206737  0.149560 -0.644941
         1  0.554534  0.540821 -1.259166  0.843940  0.397272  1.473680 -1.143226
         2  0.609446 -2.124669 -0.215270 -1.078822  1.804947 -0.177452  0.602859
         3 -0.083078 -1.665927  0.912958 -1.936269  2.581284  0.779326 -0.178385
         4  0.706565 -3.815984  1.185996  1.434728 -0.235867  0.319171  0.078276
         5  0.122691 -0.510835  1.586561  0.786459  0.638532 -0.519625  0.227352
         6  0.915105 -3.209060 -0.431365  0.697227 -1.109108  0.478450  0.737815
         7 -0.112757 -0.856601  0.471309  1.326847  1.062865 -1.247177  1.080311
         8 -1.153458 -0.188812  0.537086 -0.621523  1.715364  0.728353 -0.642681
```

9	1.996384	-1.852817	2.759792	0.862877	-0.693166	0.557047	-1.728794
10	-0.023389	-1.606689	1.097046	0.674353	1.548812	-0.501398	1.230251
11	1.060467	-2.292746	1.038405	-0.666616	1.731053	0.256155	-1.059229
12	0.492732	-1.057358	-0.200066	1.313758	0.869539	0.348615	2.096576
13	-0.149680	-2.637619	2.367622	1.113091	2.197073	-2.033489	-0.730532
14	1.022435	-4.472712	1.043839	2.302536	-1.084662	0.793423	-1.494192
15	0.371943	0.655785	1.413534	-0.940926	0.215361	2.208592	-0.546503
16	0.923958	-2.572841	-0.929223	-2.939410	0.530267	-1.746322	-0.484294
17	1.192113	-3.568228	-3.055137	0.218459	0.108713	1.433461	-1.359412
18	0.742949	-0.053752	0.696293	-0.490131	1.946102	1.169046	-0.363708
19	0.916362	-2.689798	0.299773	0.314256	0.803609	-2.572582	-0.023840
20	0.721407	-2.128203	1.963552	-0.457587	1.373763	-0.343822	-0.196675
21	0.674815	-1.705156	0.582590	-0.841685	1.724670	-0.119424	-0.625645
22	0.298448	-0.426812	0.111003	0.974969	0.735163	0.654056	-0.019187
23	1.784843	-2.962346	2.508993	1.818220	-0.507030	-0.961532	-0.527304
24	0.196452	-1.560962	1.118155	-1.339187	1.877953	1.157298	-0.505085
25	-0.046245	-1.515784	-1.919399	2.000970	-0.219451	-0.933531	1.384335
26	0.908946	-2.788528	1.131390	-0.187408	-0.419679	-0.844568	-1.662972
27	-0.594974	1.401810	-1.439898	0.288043	1.239164	1.104782	0.846444
28	-0.402196	-0.976285	-1.115387	-0.967083	1.879576	-0.263244	0.448913
29	1.295539	-1.447216	-0.483122	-0.081563	0.348007	2.196943	0.569824
..	...	...	...	...	...	...	...
248	-1.044254	2.322785	-0.259047	-0.015580	0.246615	1.354741	-0.106804
249	-2.228569	-1.591418	-2.183836	-0.368687	-0.493789	-1.254281	0.997448
250	-1.282142	-1.802805	-1.691976	-0.750117	-1.791390	-0.316083	-1.117164
251	-3.247722	1.207472	-2.683074	0.820030	1.657943	1.186589	-0.098382
252	-1.809230	-1.129631	-1.277033	-1.512849	-0.848962	-2.270452	1.793160
253	-2.583877	2.028922	1.816449	0.378805	0.169361	-1.502710	0.349765
254	-2.651169	2.935160	1.658452	-1.147751	0.521254	-0.372288	0.261536
255	-2.562416	-1.183308	-1.709678	-0.057866	-2.107534	-0.010969	-0.009147
256	-5.234386	0.834023	-2.195762	0.734344	-1.057567	-1.526783	-0.949253
257	-5.597312	4.546798	0.442118	0.939531	1.837577	-0.228771	1.057820
258	-2.810135	2.832974	0.231195	0.931447	-1.869511	1.279842	0.999190
259	-2.254607	1.692646	1.896896	0.230103	-0.181941	0.023049	1.742712
260	-2.263026	-2.374923	-1.678825	0.356506	-1.146219	0.469516	-1.880767
261	-3.893247	2.643322	1.358183	1.914464	0.943381	-0.914767	2.056901
262	-2.322565	0.344118	1.028701	-0.327519	-1.220276	-0.395270	-1.914897
263	-3.775737	2.658784	0.089879	-0.719391	-0.379839	-0.894883	0.217089
264	-1.410104	1.278028	-0.308901	2.332210	0.490535	0.762113	0.751792
265	-1.608006	1.292019	3.039977	-1.761851	-1.281105	-1.187348	1.982589
266	-2.614471	-0.974045	-2.051012	-0.433407	-1.063439	-1.128687	-0.155932
267	-2.457533	-0.349108	-4.157059	1.135841	-1.303016	1.034506	-1.997872
268	-3.596756	1.968934	-1.606319	-1.888181	-0.711893	0.091011	-1.141570
269	-3.031764	1.824557	-0.677484	0.579179	-0.089962	0.224631	1.127452
270	-2.531939	0.926230	1.854178	-1.529688	-0.861982	0.046468	0.547211
271	-1.516847	0.338494	1.116268	-0.115722	-1.769162	0.679090	-2.227469
272	-4.165657	2.610551	-0.475308	-0.078993	0.318670	0.560704	-1.581306
273	-3.567161	1.867596	-1.550686	-0.142904	-0.956396	0.716564	-1.403523

274	-3.210781	1.650820	1.003853	-0.980234	-0.955154	0.302016	-1.094609
275	-3.809440	1.992402	1.391956	-0.615517	0.203356	0.714934	0.496630
276	-4.078046	0.920309	-1.180733	0.720502	0.635725	-1.970713	-0.809786
277	-4.053575	1.632746	-1.789419	-0.404834	-0.312709	0.897971	-0.347663

	PC 8	PC 9	PC 10	target
0	0.295965	0.780602	0.496861	0
1	-1.343863	-0.277328	0.105604	0
2	-1.160820	-0.830742	-0.263305	1
3	-0.777327	-0.707167	1.102157	0
4	-1.501927	-0.935459	1.169449	1
5	-0.602075	0.305665	-0.391223	0
6	-0.008512	0.468905	-0.234740	1
7	-1.049337	-0.107014	-0.524644	0
8	-0.962033	-0.096444	1.384101	0
9	-0.214086	0.720931	-0.080748	0
10	0.274101	0.086862	-0.726848	0
11	2.087207	1.237848	0.003249	1
12	-0.295866	-1.407590	-0.237952	0
13	0.951113	-0.847759	0.557475	0
14	0.262617	1.520731	0.400775	1
15	-0.203454	-0.896631	0.180811	0
16	0.806471	0.314014	-0.807455	1
17	-0.943133	-0.649688	1.397063	1
18	-0.603527	0.137909	-0.529105	0
19	0.105423	0.460856	0.297956	1
20	0.615441	0.482376	0.289259	0
21	0.152796	-1.039225	-1.044365	0
22	-1.173677	0.444479	0.754616	1
23	-0.306868	0.621982	-0.907105	0
24	0.645305	0.042559	-0.230549	1
25	-1.654882	1.080738	0.333435	1
26	0.278137	0.302532	0.625343	1
27	-1.939746	-0.132946	0.201695	0
28	-0.840905	-1.036947	-0.555893	1
29	-0.885690	-0.139131	0.220980	1
..	...	...	...	...
248	-1.219112	-0.087364	-0.551481	0
249	1.005804	1.779031	0.333884	1
250	0.704461	0.035585	-0.172356	1
251	1.397272	-0.886407	-0.094394	0
252	-0.432595	-0.304077	0.116114	1
253	-0.863784	0.233310	-0.200168	0
254	-0.043240	1.710931	0.445044	0
255	-0.439117	1.230546	0.438291	1
256	0.040387	0.490247	-0.467562	1
257	-0.111656	0.674477	0.277142	0
258	1.196173	-0.280682	0.558777	0

```
259 -0.132027  0.199673 -0.925812       0
260  0.553541 -0.669197  0.044001       1
261  0.795963 -0.372068  0.560166       0
262  0.896235  0.320924 -0.256060       0
263 -0.777229 -0.048127 -0.606024       0
264  0.668129  0.002507 -1.494993       0
265 -0.695438  0.224201  0.725232       0
266 -0.329988  0.362460  0.192491       1
267 -0.379441  0.073311 -1.589330       1
268 -0.484582 -1.035115  1.685102       1
269  0.451602  1.925917 -0.215019       0
270  0.327784 -0.200252  0.730195       0
271 -0.392132  0.682422 -0.144976       0
272 -0.418458  0.307544  0.635986       0
273 -0.013750 -0.307083 -0.408206       1
274 -0.213238  0.215082  0.260386       0
275 -0.925259 -1.727557 -0.169179       0
276 -0.900832  0.132112  0.677522       1
277  1.782907 -1.158332  0.231105       1

[278 rows x 11 columns]
```

Before we plot the principal components, lets take a look at the variance explained by the principal components

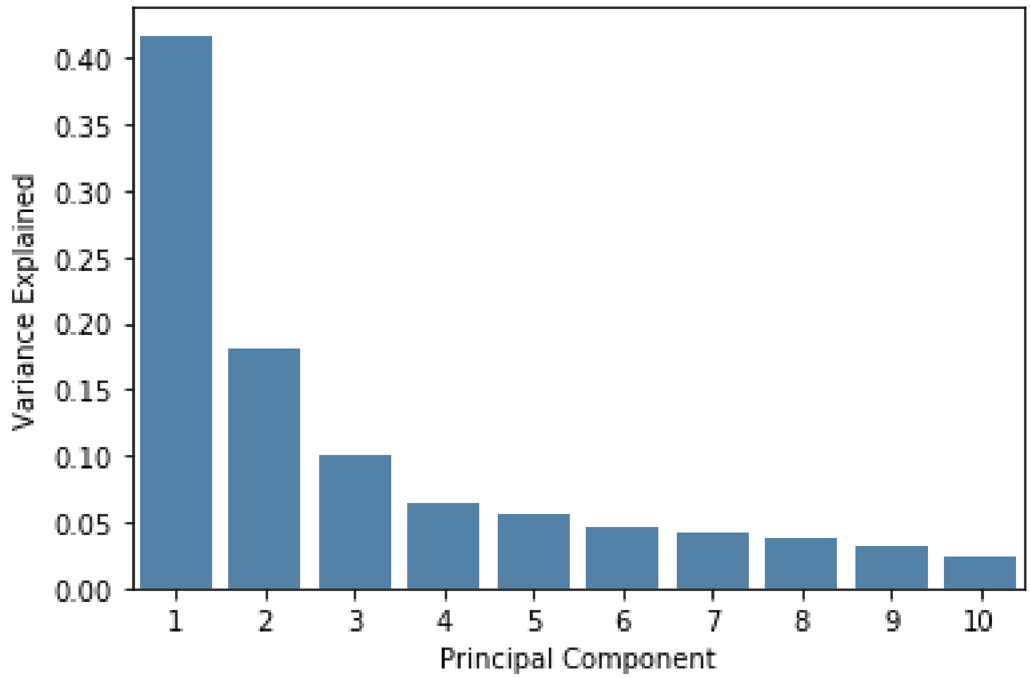
In [43]:
```
variance=np.var(principal_components,axis=0)
variance_ratio = variance/np.sum(variance)
component_no = [1,2,3,4,5,6,7,8,9,10]
print(variance_ratio)
```

```
[0.417515   0.18038884 0.10079889 0.06489423 0.05699299 0.04632148
 0.04169407 0.03711056 0.0314276  0.02285634]
```

Looking at the percent variance explained by each principle component, you can see that the first component explains 69.8% of the variation. Meanwhile, the second component explains 30.1% of the variation. The scree plot below demonstrates this.

In [44]:
```
sns.barplot(component_no,variance_ratio, color = 'steelblue').set(xlabel = "Principal
```

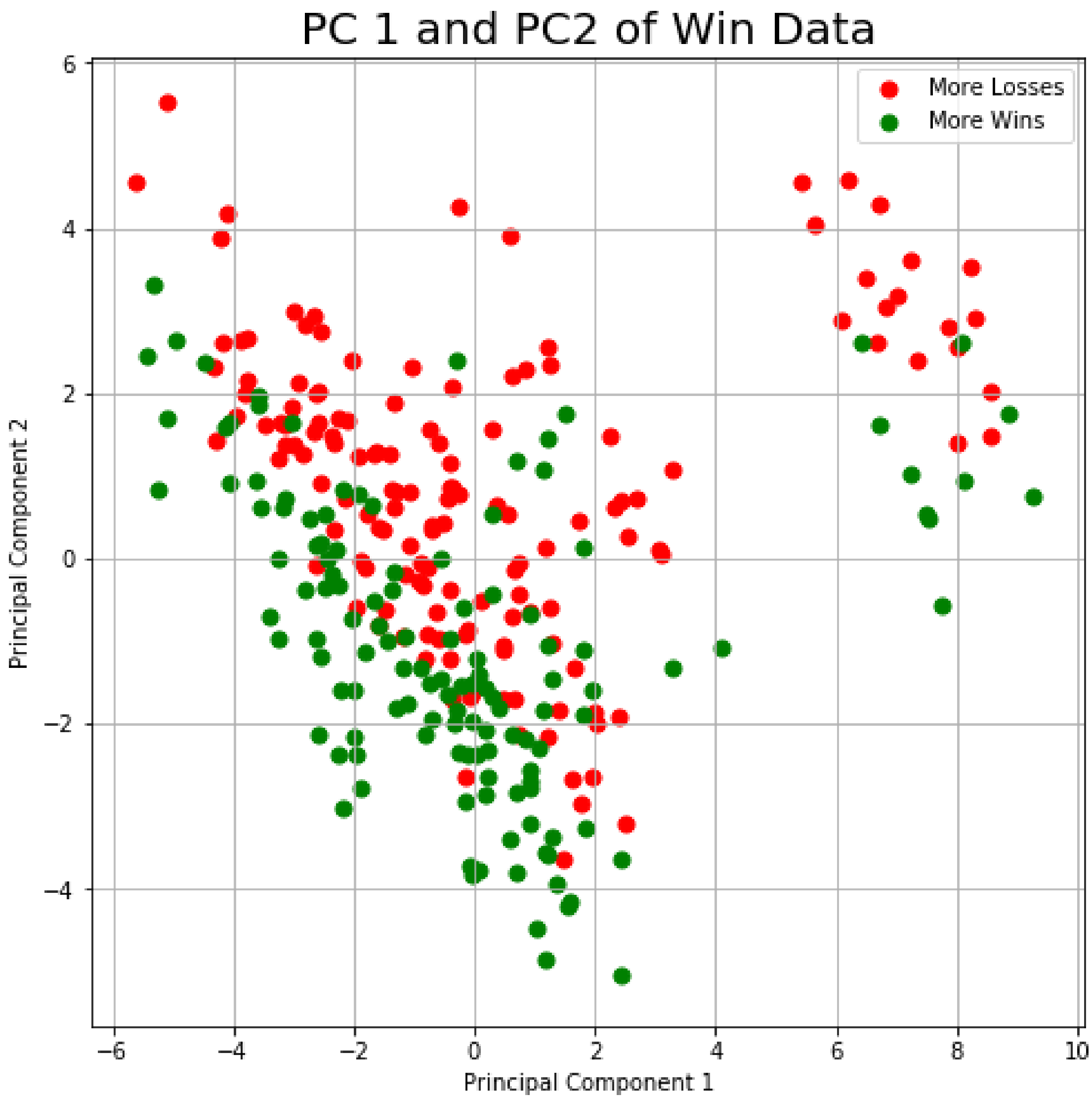
Out[44]:
```
[Text(0, 0.5, 'Variance Explained'), Text(0.5, 0, 'Principal Component')]
```



Now, we can use the pc_df to visualize the principal components.

```
In [45]: fig = plt.figure(figsize = (8,8))
         ax = fig.add_subplot(1,1,1)
         ax.set_xlabel('Principal Component 1')
         ax.set_ylabel('Principal Component 2')
         ax.set_title('PC 1 and PC2 of Win Data', fontsize = 20)
         targets = [0,1]
         colors = ['r', 'g']
         for target, color in zip(targets,colors):
             indicesToKeep = pc_df['target'] == target
             ax.scatter(pc_df.loc[indicesToKeep, 'PC 1']
                        , pc_df.loc[indicesToKeep, 'PC 2']
                        , c = color
                        , s = 50)
         ax.legend(targets,labels = ['More Losses', 'More Wins'])
         ax.grid()
```

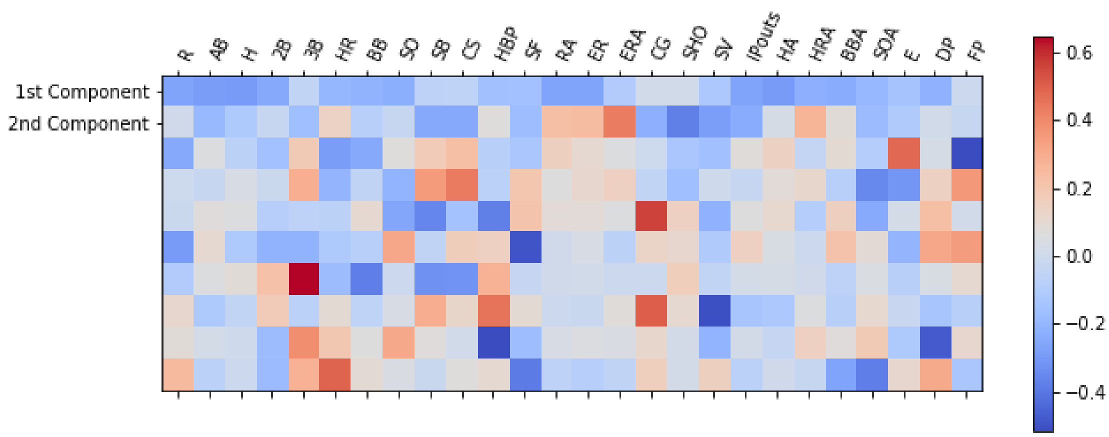
```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:14: UserWarning: You have mixed p
```



Observing the graph of the two principal components, there is a clear distinguishing of teams with more wins and teams with more losses on the line $y = -x - 2$. On the bottom portion of the line, there seems to be teams with more wins. Above the line there are more teams that have more losses. There is another grouping of points in the upper right corner from the bigger grouping of points. The extremely interesting part about this is that the groupings of points are in the same layout at the bigger grouping. The line $y = \frac{5}{6}x + \frac{25}{3}$ divides the teams with more wins below the line, while the teams above the line are mostly teams with more losses. After some investigation, there were four expansion teams in the 1990's: Colorado Rockies(1993), Florida Marlins(1993) (Now Miami Marlins), Arizona Diamondbacks(1998), and Tampa Bay Devil Rays(1998) (Now Tampa Bay Rays). It is possible that because the analyzed data was from 1990 to 2000, this grouping represents these teams. Unfortunately, it is impossible to definitely say based on this visualization.

```
In [46]: plt.matshow(pca.components_,cmap='coolwarm')
         plt.yticks([0,1],['1st Component','2nd Component'],fontsize=10)
```

```
plt.colorbar()
plt.xticks(range(len(teams_90.loc[:,'R':'FP'].columns)),teams_90.loc[:,'R':'FP'].colu
plt.show()
```



The plot above shows how much of each variable accounted for each principal component. Noticeably, the second component seems to have been based highly off or Earned Run Average (ERA), Run Average (RA), Earned Runs (ER), Home Runs (HR), and Home Runs Allowed (HRA).

The ratio of runs to earned runs seems to be a promising for a regression of Fractional wins on R/ER

1.4 Conclusions and Future Work (200 words)

In this section you should summarize your findings based on your final model in clearly understandable, non-statistical terms. What is the main message produced by your analysis? There may also be additional questions that arise, problems you encounter, or possible extensions of your analysis that could be addressed here.

Include any final comments and thoughts about your project. For example, do you trust your results? How general are your results, to what situations do they apply? Add any other comments that are relevant.

1.5 Grading Criteria

Your grade on the project will be based on the following criteria:

1. Compatibility of Scientific Question and Analysis

Is the scientific question being addressed actually of interest, and were suitable tools employed?

2. Coherent Thought Process and Presentation of Results

- Does the analysis indicate a sound understanding of methods discussed in class? This is often best judged by the preliminary comments on the questions of interest as well as the conclusions made after the analysis.

- Is the analysis presented in a clear, consistent, coherent style with the appropriately labeled requested components and visualizations?

3. Scope of the analysis and methods used
Did the analysis demonstrate a wide understanding of methodology and ideas presented throughout the quarter?

4. Reproducible results
Is it possible to reproduce the analysis and the visuals by executing the provided code?

1.6 Submission

You are required to submit two files:

1. Submit your completed writeup as a PDF to gradescope. You should address all of the components described above, adhering to the page limit, and include any figures and tables that are necessary. (Make sure to number figures and tables and include informative captions.)

2. Submit a complete jupyter notebook with all of your analyses to the okpy server. For your submission, use **this jupyter notebook** as a template (remove the instructions, replacing them with your analysis). We should be able to reproduce all of your results by running your notebook.

Before you submit the notebook, make sure that you select from the top menu `Kernel -> Restart & Clear Output` followed by `Cell -> Run All`. Verify that all computations execute correctly. There should be no errors when we run your notebook.

```
In [47]: # These lines help load your submission for grading.
         from client.api.notebook import Notebook
         ok = Notebook('final-project.ok')
         _ = ok.auth(inline=True)
```

```
=====================================================================
Assignment: final-project
OK, version v1.14.15
=====================================================================

Successfully logged in as matthewbcoleman@ucsb.edu
```

```
In [ ]: _ = ok.submit()
```

```
<IPython.core.display.Javascript object>
```

```
In [ ]:
```