NBA Analysis

June 24, 2020

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

[2]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

1 Introduction

In this notebook we will examine the Basketball datasets, and derive relationships between college player performance and success in the NBA. Specifically, we will see if NCAA data can help coaches predict how well a player might perofrm in terms of points per game, 3 point percentage, field goal percentage, and free throw percentage

[3]:
```python
official_box_score = pd.read_csv("2012-18_officialBoxScore.csv")
team_box_score =  pd.read_csv("2012-18_teamBoxScore.csv")
player_box_score = pd.read_csv("2012-18_playerBoxScore.csv")
standings = pd.read_csv("2012-18_standings.csv")
college = pd.read_csv("college.csv")
```

[4]:
```python
official_box_score.head()
```

[4]:
```
      gmDate gmTime  seasTyp    offLNm    offFNm teamAbbr teamConf  \
0  2012-10-30  19:00  Regular  Brothers      Tony      WAS     East
1  2012-10-30  19:00  Regular     Smith   Michael      WAS     East
2  2012-10-30  19:00  Regular   Workman  Haywoode      WAS     East
3  2012-10-30  19:00  Regular  Brothers      Tony      CLE     East
4  2012-10-30  19:00  Regular     Smith   Michael      CLE     East

     teamDiv teamLoc teamRslt  …  opptFIC40  opptOrtg  opptDrtg  opptEDiff  \
0  Southeast    Away     Loss  …    61.6667  105.6882   94.4447    11.2435
1  Southeast    Away     Loss  …    61.6667  105.6882   94.4447    11.2435
2  Southeast    Away     Loss  …    61.6667  105.6882   94.4447    11.2435
3    Central    Home      Win  …    56.0417   94.4447  105.6882   -11.2435
4    Central    Home      Win  …    56.0417   94.4447  105.6882   -11.2435

   opptPlay%   opptAR  opptAST/TO  opptSTL/TO     poss     pace
```

```
0     0.4390  16.7072       1.0476     33.3333  88.9409  88.9409
1     0.4390  16.7072       1.0476     33.3333  88.9409  88.9409
2     0.4390  16.7072       1.0476     33.3333  88.9409  88.9409
3     0.3765  18.8679       2.0000     84.6154  88.9409  88.9409
4     0.3765  18.8679       2.0000     84.6154  88.9409  88.9409

[5 rows x 119 columns]
```

[5]:
```
team_box_score.head()
```

[5]:
```
       gmDate gmTime  seasTyp    offLNm1 offFNm1    offLNm2  offFNm2  \
0  2012-10-30  19:00  Regular   Brothers    Tony      Smith  Michael
1  2012-10-30  19:00  Regular   Brothers    Tony      Smith  Michael
2  2012-10-30  20:00  Regular  McCutchen   Monty     Wright     Sean
3  2012-10-30  20:00  Regular  McCutchen   Monty     Wright     Sean
4  2012-10-30  22:30  Regular     Foster   Scott  Zielinski     Gary

      offLNm3   offFNm3 teamAbbr  … opptFIC40  opptOrtg  opptDrtg opptEDiff  \
0     Workman  Haywoode      WAS  …   61.6667  105.6882   94.4447   11.2435
1     Workman  Haywoode      CLE  …   56.0417   94.4447  105.6882  -11.2435
2  Fitzgerald      Kane      BOS  …   80.8333  126.3381  112.6515   13.6866
3  Fitzgerald      Kane      MIA  …   62.7083  112.6515  126.3381  -13.6866
4       Dalen      Eric      DAL  …   58.6458   99.3678  108.1034   -8.7356

  opptPlay%   opptAR  opptAST/TO  opptSTL/TO     poss     pace
0     0.4390  16.7072       1.0476     33.3333  88.9409  88.9409
1     0.3765  18.8679       2.0000     84.6154  88.9409  88.9409
2     0.5244  19.8287       3.1250    100.0000  94.9832  94.9832
3     0.4643  18.8501       1.5000     25.0000  94.9832  94.9832
4     0.5000  18.6567       1.7143     42.8571  91.5790  91.5790

[5 rows x 123 columns]
```

[6]:
```
player_box_score.head()
```

[6]:
```
       gmDate gmTime  seasTyp playLNm  playFNm teamAbbr teamConf   teamDiv  \
0  2012-10-30  19:00  Regular   Price     A.J.      WAS     East  Southeast
1  2012-10-30  19:00  Regular   Ariza   Trevor      WAS     East  Southeast
2  2012-10-30  19:00  Regular  Okafor    Emeka      WAS     East  Southeast
3  2012-10-30  19:00  Regular    Beal  Bradley      WAS     East  Southeast
4  2012-10-30  19:00  Regular  Booker   Trevor      WAS     East  Southeast

  teamLoc teamRslt  …  playFT% playORB playDRB playTRB opptAbbr opptConf  \
0    Away     Loss  …      1.0       1       1       2      CLE     East
1    Away     Loss  …      0.5       1       2       3      CLE     East
2    Away     Loss  …      0.5       5       2       7      CLE     East
3    Away     Loss  …      1.0       0       3       3      CLE     East
```

```
4    Away     Loss  …      0.0       1       0       1      CLE      East

   opptDiv opptLoc opptRslt  opptDayOff
0  Central    Home      Win           0
1  Central    Home      Win           0
2  Central    Home      Win           0
3  Central    Home      Win           0
4  Central    Home      Win           0

[5 rows x 51 columns]
```

[7]:
```
standings.head()
```

[7]:
```
       stDate teamAbbr  rank rankOrd  gameWon  gameLost stk stkType  stkTot  \
0  2012-10-30      ATL     3     3rd        0         0   -       -       0
1  2012-10-30      BKN     3     3rd        0         0   -       -       0
2  2012-10-30      BOS    14    14th        0         1  L1    loss       1
3  2012-10-30      CHA     3     3rd        0         0   -       -       0
4  2012-10-30      CHI     3     3rd        0         0   -       -       0

   gameBack  …  rel%Indx   mov   srs    pw%  pyth%13.91  wpyth13.91  \
0       0.5  …       0.0   0.0   0.0  0.500      0.0000      0.0000
1       0.5  …       0.0   0.0   0.0  0.500      0.0000      0.0000
2       1.0  …       0.0 -13.0 -13.0  0.072      0.1687     13.8334
3       0.5  …       0.0   0.0   0.0  0.500      0.0000      0.0000
4       0.5  …       0.0   0.0   0.0  0.500      0.0000      0.0000

   lpyth13.91  pyth%16.5  wpyth16.5  lpyth16.5
0     82.0000      0.000      0.000     82.000
1     82.0000      0.000      0.000     82.000
2     68.1666      0.131     10.742     71.258
3     82.0000      0.000      0.000     82.000
4     82.0000      0.000      0.000     82.000

[5 rows x 39 columns]
```

[8]:
```
college.head()
```

[8]:
```
   Unnamed: 0  active_from  active_to        birth_date  \
0           0         1991       1995     June 24, 1968
1           1         1969       1978     April 7, 1946
2           2         1970       1989    April 16, 1947
3           3         1991       2001     March 9, 1969
4           4         1998       2003  November 3, 1974

                                     college height  \
0                              Duke University   6-10
```

```
1                        Iowa State University    6-9
2           University of California, Los Angeles    7-2
3                     Louisiana State University    6-1
4  University of Michigan, San Jose State University    6-6

                name position                       url  weight  …  \
0     Alaa Abdelnaby      F-C  /players/a/abdelal01.html   240.0  …
1     Zaid Abdul-Aziz      C-F  /players/a/abdulza01.html   235.0  …
2  Kareem Abdul-Jabbar        C  /players/a/abdulka01.html   225.0  …
3   Mahmoud Abdul-Rauf        G  /players/a/abdulma02.html   162.0  …
4    Tariq Abdul-Wahad        F  /players/a/abdulta01.html   223.0  …

   NCAA__3ptpg  NCAA_efgpct  NCAA_fgapg  NCAA_fgpct  NCAA_fgpg  NCAA_ft  \
0          0.0          NaN         5.6       0.599        3.3    0.728
1          NaN          NaN         NaN         NaN        NaN      NaN
2          NaN          NaN        16.8       0.639       10.7    0.628
3          2.7          NaN        21.9       0.474       10.4    0.863
4          NaN          NaN         NaN         NaN        NaN      NaN

   NCAA_ftapg  NCAA_ftpg  NCAA_games  NCAA_ppg
0         2.5        1.8       134.0       8.5
1         NaN        NaN         NaN       NaN
2         7.9        5.0        88.0      26.4
3         6.4        5.5        64.0      29.0
4         NaN        NaN         NaN       NaN

[5 rows x 34 columns]
```

1.1 Part 1: Data Cleaning

For determing how college determines NBA success, we will combine the college dataset with nba statistics that we care about, joining on the name of the player. First, let us remove the columns from the college dataset with a lot of null values. We will also drop unncessary columns

[9]:
```
college.shape
```

[9]:
```
(4576, 34)
```

We see that the college dataset has 4576 players, and has 34 attributes for each player. Let's see which of these columns has many null values

[10]:
```
college.isna().mean()
```

[10]:
```
Unnamed: 0          0.000000
active_from         0.000000
active_to           0.000000
birth_date          0.006337
```

```
college              0.065997
height               0.000219
name                 0.000000
position             0.000219
url                  0.000000
weight               0.001311
NBA__3ptapg          0.246503
NBA__3ptpct          0.354677
NBA__3ptpg           0.246503
NBA_efgpct           0.251311
NBA_fg%              0.006119
NBA_fg_per_game      0.000000
NBA_fga_per_game     0.000000
NBA_ft%              0.043269
NBA_ft_per_g         0.000000
NBA_fta_p_g          0.000000
NBA_g_played         0.000000
NBA_ppg              0.000000
NCAA__3ptapg         0.591783
NCAA__3ptpct         0.622815
NCAA__3ptpg          0.591128
NCAA_efgpct          1.000000
NCAA_fgapg           0.435752
NCAA_fgpct           0.435533
NCAA_fgpg            0.432255
NCAA_ft              0.433129
NCAA_ftapg           0.433566
NCAA_ftpg            0.432255
NCAA_games           0.432255
NCAA_ppg             0.432255
dtype: float64
```

We see that a good amount of these columns have many null values, so let's remove the players that don't have the NCAA information that we care about, as well as columns that are significantly empty

[11]:
```
college = college[college.columns[college.isnull().mean() < 0.5]]
```

The player college and college states is are most important featurse that we care about, so let's remove all rows that don't have a that information. Also let's remove some unnessecary columns.

[12]:
```
college.dropna(subset=['college','position', 'NCAA_fgapg',␣
 ↪'NCAA_fgpct',"NBA__3ptapg"], inplace=True)
college.drop(columns=['url', 'birth_date', 'Unnamed: 0', "active_from",␣
 ↪"active_to"], inplace=True)
```

[13]:
```
college.isna().sum()
```

[13]:
```
college              0
height               0
name                 0
position             0
weight               0
NBA__3ptapg          0
NBA__3ptpct        310
NBA__3ptpg           0
NBA_efgpct          15
NBA_fg%             15
NBA_fg_per_game      0
NBA_fga_per_game     0
NBA_ft%            111
NBA_ft_per_g         0
NBA_fta_p_g          0
NBA_g_played         0
NBA_ppg              0
NCAA_fgapg           0
NCAA_fgpct           0
NCAA_fgpg            0
NCAA_ft              0
NCAA_ftapg           0
NCAA_ftpg            0
NCAA_games           0
NCAA_ppg             0
dtype: int64
```

Let's fill the null quantitative values with the average value of the column

[14]:
```
college.fillna(college.mean(), inplace=True)
```

We have now fully cleaned up the data, and can now proceed with analysis of the data

1.2 Part 2: Exploratory Data Analysis/Additional Cleaning

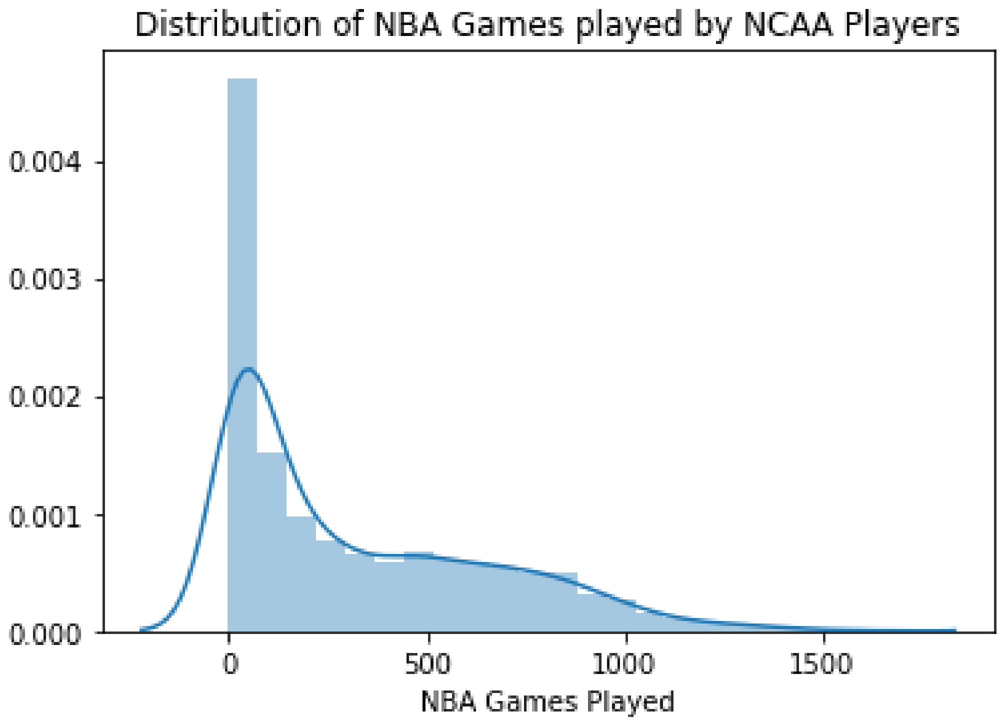
[15]:
```
print(college.shape)
```

```
(2536, 25)
```

We have 2536 observations, with 25 columns. First, let's see how many games most of these players play in the NBA

[16]:
```
sns.distplot(college['NBA_g_played'], axlabel= "NBA Games Played").
 ↪set_title('Distribution of NBA Games played by NCAA Players')
```

[16]:
```
Text(0.5, 1.0, 'Distribution of NBA Games played by NCAA Players')
```



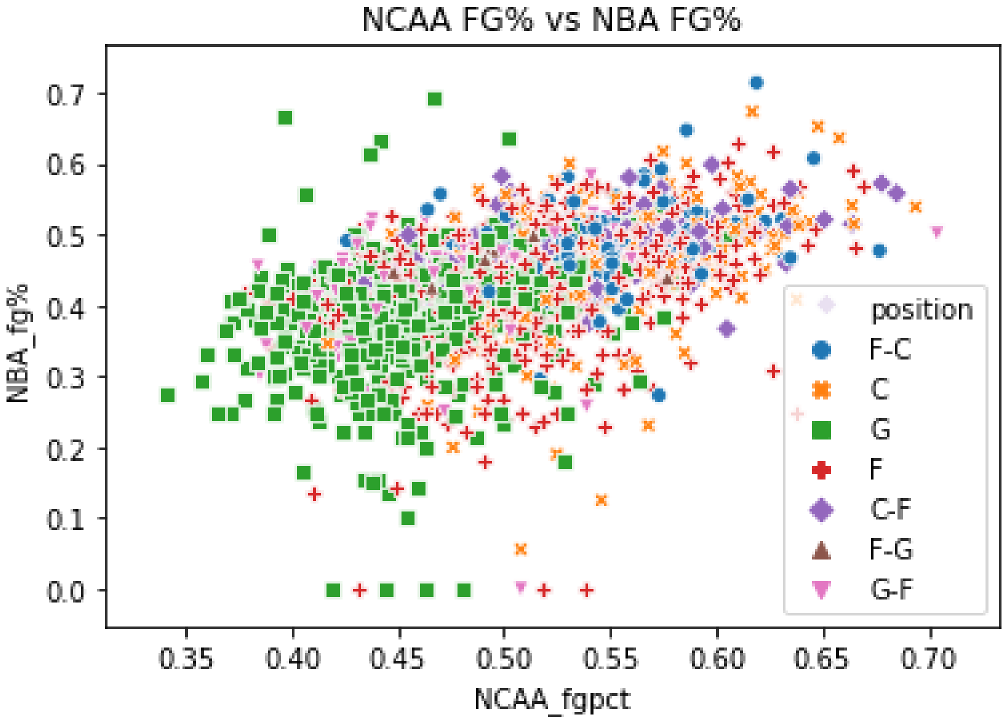
It looks like some of these players never make it to the NBA/never get to play NBA games, so lets only keep the players that played at least 5 games

[17]:
```
college = college.loc[college['NBA_g_played'] >=5]
college.shape
```

[17]: (2403, 25)

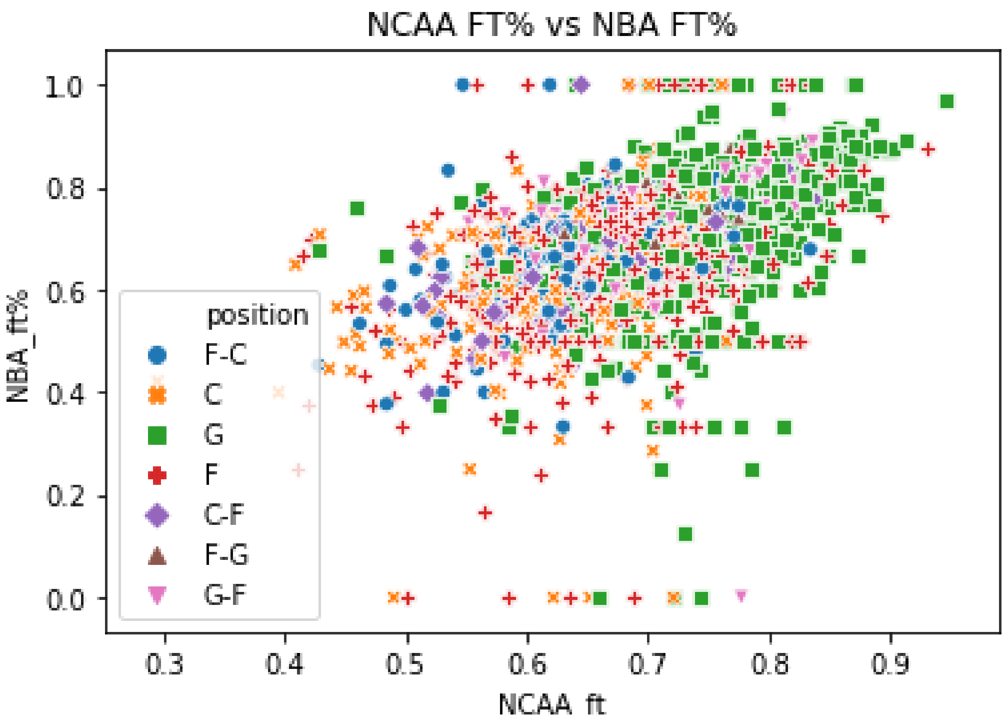
Let's see if there's a good relationship between field goal and free throw percentage, as well as points per game in the NCAA and NBA. Unfortunately, we can't use the NCAA 3 point data since there was so little of it

[18]:
```
ax=sns.scatterplot(x="NCAA_fgpct", y="NBA_fg%",hue= 'position',␣
 ↪style='position', data=college).set_title('NCAA FG% vs NBA FG%')
```



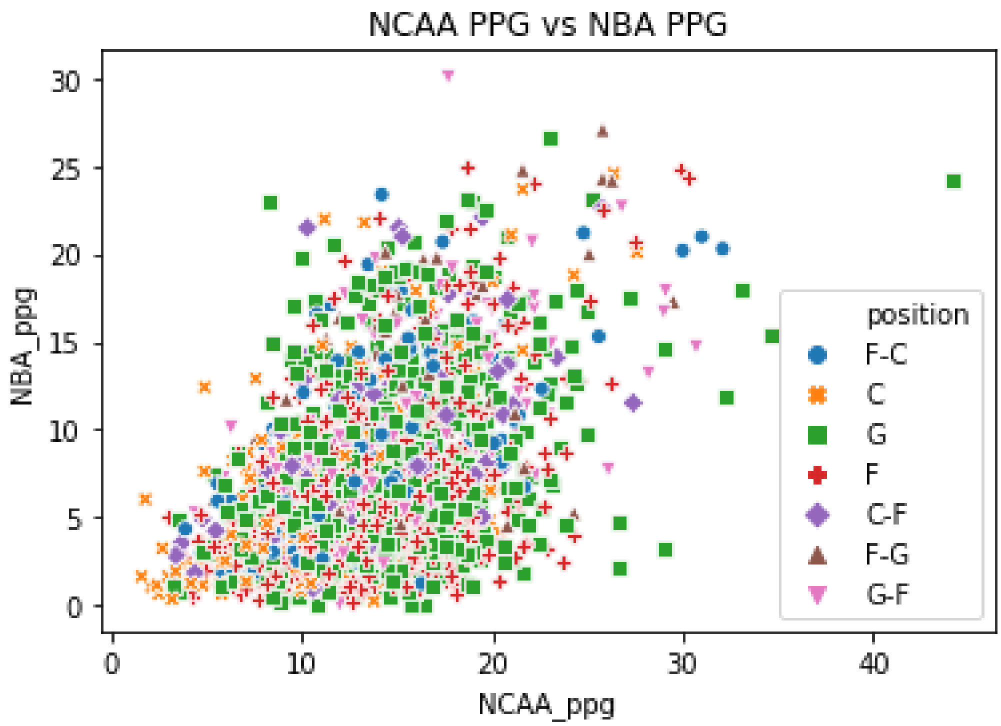
[19]:
```
sns.scatterplot(x="NCAA_ft", y="NBA_ft%",hue= 'position', style='position',␣
↪data=college).set_title('NCAA FT% vs NBA FT%')
```

[19]: Text(0.5, 1.0, 'NCAA FT% vs NBA FT%')



[20]:
```
sns.scatterplot(x="NCAA_ppg", y="NBA_ppg",hue= 'position', style='position',␣
↪data=college).set_title('NCAA PPG vs NBA PPG')
```

[20]: Text(0.5, 1.0, 'NCAA PPG vs NBA PPG')



There seems to be a relatively linear relationship between all of these variables, and position also appears to play a nontrivial role in both field goal percentage and free throw percentage. These relationships might be more evident if we center and standardize the data.

[21]:
```
college.college.nunique()
```

[21]:
```
448
```

The college column has 448 unique values, so it might be hard to conduct analysis with it using one-hot encoding. We'll keep it for now, and remove it later if it presents an issue

Side note: one player appears to have been absolutely dominating in both the college level and the NBA in points per game, so lets quickly see who that is

[22]:
```
college[college['NCAA_ppg']>40]
```

[22]:
```
                       college height          name position  weight  \
2539  Louisiana State University    6-5  Pete Maravich        G   197.0

      NBA__3ptapg  NBA__3ptpct  NBA__3ptpg  NBA_efgpct  NBA_fg%  …  \
2539          0.3        0.667         0.2       0.441    0.441  …

      NBA_g_played  NBA_ppg  NCAA_fgapg  NCAA_fgpct  NCAA_fgpg  NCAA_ft  \
2539           658     24.2        38.1       0.438       16.7    0.775
```

```
      NCAA_ftapg  NCAA_ftpg  NCAA_games  NCAA_ppg
2539        13.9       10.8        83.0      44.2

[1 rows x 25 columns]
```

The player is Pistol Pete, one of the greatest offensive players of all time, who was active in the 1970s. Also, that pink triangle at the top of the ppg plot is Michael Jordan, who went from less than 20 points per game in college to over 30 in the NBA

1.3 Part 3: Transformations/Feature Engineering

As we concluded in the previous section, it may be useful to center and standardize the data, so we will do that now. Additonally, we will make the categorical features useful with one-hot encoding. First, lets convert the height column so that it is an int value, in inches.

[23]:
```
college["height"] = college["height"].str[:1].astype(int)*12 +␣
↪college["height"].str[2:].astype(int)
college
```

[23]:
```
                                                college  height  \
0                                       Duke University      82
2                      University of California, Los Angeles      86
3                                Louisiana State University      73
5                                  University of California      81
6                                        Indiana University      79
…                                                   …     …
4564                               Arizona State University      80
4565  Seattle Pacific University, University of Wash…      82
4567                 University of California, Los Angeles      84
4569                      University of Nevada, Las Vegas      84
4572                                  Kent State University      85

                    name position  weight  NBA__3ptapg  NBA__3ptpct  \
0         Alaa Abdelnaby      F-C   240.0          0.0     0.000000
2      Kareem Abdul-Jabbar        C   225.0          0.0     0.056000
3       Mahmoud Abdul-Rauf        G   162.0          2.3     0.354000
5      Shareef Abdur-Rahim        F   225.0          0.6     0.297000
6            Tom Abernethy        F   220.0          0.0     0.000000
…                    …      …     …          …          …
4564             Tony Zeno        F   210.0          0.0     0.240532
4565      Phil Zevenbergen        C   230.0          0.0     0.240532
4567          George Zidek        C   250.0          0.0     0.250000
4569    Stephen Zimmerman        C   240.0          0.0     0.240532
4572              Jim Zoet        C   240.0          0.0     0.240532

      NBA__3ptpg  NBA_efgpct  NBA_fg%  …  NBA_g_played  NBA_ppg  NCAA_fgapg  \
0            0.0       0.502    0.502  …           256      5.7         5.6
```

```
2              0.0        0.559     0.559  …          1560      24.6          16.8
3              0.8        0.472     0.442  …           586      14.6          21.9
5              0.2        0.479     0.472  …           830      18.1          14.2
6              0.0        0.492     0.492  …           319       5.6           4.5
…          …          …      …  …           …      …         …
4564           0.0        0.286     0.286  …             8       1.8          10.4
4565           0.0        0.556     0.556  …             8       3.8           7.6
4567           0.0        0.409     0.408  …           135       3.4           5.4
4569           0.0        0.323     0.323  …            19       1.2           8.2
4572           0.0        0.200     0.200  …             7       0.3           2.9

      NCAA_fgpct  NCAA_fgpg  NCAA_ft  NCAA_ftapg  NCAA_ftpg  NCAA_games  \
0          0.599        3.3    0.728         2.5        1.8       134.0
2          0.639       10.7    0.628         7.9        5.0        88.0
3          0.474       10.4    0.863         6.4        5.5        64.0
5          0.518        7.4    0.683         8.9        6.1        28.0
6          0.533        2.4    0.689         1.7        1.1       110.0
…            …      …      …          …     …          …
4564       0.466        4.8    0.742         2.4        1.8       112.0
4565       0.501        3.8    0.721         3.5        2.5        66.0
4567       0.520        2.8    0.744         2.1        1.5       104.0
4569       0.477        3.9    0.624         3.9        2.4        26.0
4572       0.476        1.4    0.429         1.0        0.4        63.0

      NCAA_ppg
0          8.5
2         26.4
3         29.0
5         21.1
6          5.9
…         …
4564      11.4
4565      10.1
4567       7.1
4569      10.5
4572       3.2

[2403 rows x 25 columns]
```

Lets extract the qualitative values, and center and standardize them

[24]:
```
college_quant = college.select_dtypes(include=['number'])
mean = np.mean(college_quant)
std = np.std(college_quant)
college_cent_stand = (college_quant - mean)/std
college_cent_stand
```

```
[24]:        height    weight  NBA__3ptapg  NBA__3ptpct  NBA__3ptpg  NBA_efgpct  \
      0     0.981752  1.027288    -0.770034    -1.697539   -0.691268    0.607731
      2     2.121852  0.452095    -0.770034    -1.303024   -0.691268    1.397792
      3    -1.583471 -1.963713     1.081806     0.796354    1.049717    0.191910
      5     0.696727  0.452095    -0.286945     0.394795   -0.256022    0.288935
      6     0.126678  0.260365    -0.770034    -1.697539   -0.691268    0.469124
      …         …         …           …          …           …           …
      4564  0.411703 -0.123097    -0.770034    -0.003017   -0.691268   -2.386183
      4565  0.981752  0.643826    -0.770034    -0.003017   -0.691268    1.356210
      4567  1.551802  1.410750    -0.770034     0.063685   -0.691268   -0.681315
      4569  1.551802  1.027288    -0.770034    -0.003017   -0.691268   -1.873336
      4572  1.836827  1.027288    -0.770034    -0.003017   -0.691268   -3.578204

             NBA_fg%  NBA_fg_per_game  NBA_fga_per_game   NBA_ft%  …  \
      0     0.943011        -0.127966         -0.283998 -0.079400  …
      2     1.722096         4.066708          3.169393  0.078526  …
      3     0.122922         1.808037          2.000952  1.531447  …
      5     0.532967         2.076927          2.078848  0.781298  …
      6     0.806330        -0.181744         -0.335929  0.283830  …
      …         …               …                 …         …  …
      4564 -2.009309        -0.988412         -0.855236  2.281597  …
      4565  1.681091        -0.396855         -0.647513 -5.614715  …
      4567 -0.341795        -0.773300         -0.777340  0.568097  …
      4569 -1.503587        -1.149745         -1.114889 -0.876928  …
      4572 -3.184769        -1.364857         -1.348577 -0.027382  …

            NBA_g_played   NBA_ppg  NCAA_fgapg  NCAA_fgpct  NCAA_fgpg   NCAA_ft  \
      0        -0.260091 -0.241803   -1.325867    1.826610  -1.014274  0.194507
      2         3.663086  3.641613    1.844172    2.538597   3.229696 -1.007904
      3         0.732738  1.586895    3.287672   -0.398348   3.057643  1.817763
      5         1.466829  2.306046    1.108270    0.384837   1.337115 -0.346578
      6        -0.070551 -0.262351   -1.637211    0.651832  -1.530432 -0.274433
      …              …         …           …           …          …         …
      4564     -1.006216 -1.043143    0.032721   -0.540745  -0.154010  0.362845
      4565     -1.006216 -0.632200   -0.759789    0.082243  -0.727519  0.110339
      4567     -0.624128 -0.714388   -1.382475    0.420437  -1.301029  0.386893
      4569     -0.973122 -1.166426   -0.589965   -0.344949  -0.670168 -1.056001
      4572     -1.009225 -1.351351   -2.090073   -0.362749  -2.103942 -3.400704

            NCAA_ftapg  NCAA_ftpg  NCAA_games  NCAA_ppg
      0      -1.008474  -0.932307    1.165570 -1.149654
      2       2.541278   1.813305   -0.352263  2.866955
      3       1.555236   2.242307   -1.144176  3.450373
      5       3.198640   2.757110   -2.332046  1.677679
      6      -1.534363  -1.532910    0.373657 -1.733072
      …           …          …           …         …
      4564   -1.074210  -0.932307    0.439650 -0.498919
```

```
4565   -0.351113  -0.331704   -1.078184 -0.790628
4567   -1.271419  -1.189708    0.175679 -1.463802
4569   -0.088168  -0.417505   -2.398039 -0.700871
4572   -1.994516  -2.133513   -1.177173 -2.338929

[2403 rows x 22 columns]
```

Now lets add back position and college, using one-hot encoding

[25]:
```
coll = pd.get_dummies(college.college, prefix='school')
pos = pd.get_dummies(college.position, prefix='pos')
college_cent_stand = college_cent_stand.join(pos)
college_cent_stand =college_cent_stand.join(coll)
```

Now that our data is ready, it is time to test out some prediction models

1.4 Part 4: Inference

[26]:
```
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor

def rmse_score(model, X, y):
    return np.sqrt(np.mean((y - model.predict(X))**2))
```

First, lets split the dataframe into the our X and Y values, and create a train test split

[27]:
```
Y = college_cent_stand[["NBA__3ptapg",␣
 ↪"NBA__3ptpct","NBA__3ptpg","NBA_efgpct","NBA_fg%","NBA_fg_per_game",
                        "NBA_fga_per_game",'NBA_ft%', 'NBA_ft_per_g',␣
 ↪'NBA_fta_p_g', 'NBA_g_played', 'NBA_ppg']]
X = college_cent_stand[college_cent_stand.columns.difference(["NBA__3ptapg",␣
 ↪"NBA__3ptpct","NBA__3ptpg","NBA_efgpct","NBA_fg%","NBA_fg_per_game",
                        "NBA_fga_per_game",'NBA_ft%', 'NBA_ft_per_g',␣
 ↪'NBA_fta_p_g', 'NBA_g_played', 'NBA_ppg'])]
```

For the sake of brevity, we will determine ideal model based on minimal loss on NBA field goal percentage, and predict other features using the model we find

[28]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y["NBA_fg%"],␣
 ↪test_size=0.2)
```

Lets first see how a basic linear model does

[29]:
```
linear = linear_model.LinearRegression()
linear.fit(X_train,Y_train)
```

```
np.mean(cross_val_score(linear, X_train, Y_train, scoring=rmse_score, cv=5))
```

[29]: 244922280115.26846

The linear model performs incredibly poorly here, let's see what happens if we use Ridge and Lasso Regularization

[30]:
```
alphas = np.linspace(0.1, 3, 30)
ridge = linear_model.RidgeCV(alphas=alphas)
ridge.fit(X_train,Y_train)
np.mean(cross_val_score(ridge, X_train, Y_train, scoring=rmse_score, cv=5))
```

[30]: 0.8978736882161789

[31]:
```
lasso = linear_model.LassoCV(alphas=alphas)
lasso.fit(X_train,Y_train)
np.mean(cross_val_score(lasso, X_train, Y_train, scoring=rmse_score, cv=5))
```

[31]: 0.9038790296345743

[32]:
```
elastic = linear_model.ElasticNetCV(alphas=alphas)
elastic.fit(X_train,Y_train)
np.mean(cross_val_score(elastic, X_train, Y_train, scoring=rmse_score, cv=5))
```

[32]: 0.8953872982882174

We see a significant improvement in cross validation score with these three models. This makes sense as our training data has 435 columns, so some regularization is needed to prevent overfitting. Let's also check how a random forest performs with this data

[33]:
```
forest = RandomForestRegressor()
forest.fit(X_train,Y_train)
np.mean(cross_val_score(forest, X_train, Y_train, scoring=rmse_score, cv=5))
```

[33]: 0.9254525295960592

While not as good as linear models with regularization, the random forest performs at a similar level, and much better than a standard linear model. Lets plot some comparisons of all these models. (We'll exclude the linear model since it was so bad, and would ruin the graph)

[34]:
```
models = {}
models["RidgeCV"] = ridge
models["LassoCV"] = lasso
models["ElasticCV"] = elastic
models["RandomForest"] = forest
```

[35]:
```
#Adapted from Lecture 19 code
import plotly.graph_objects as go
```

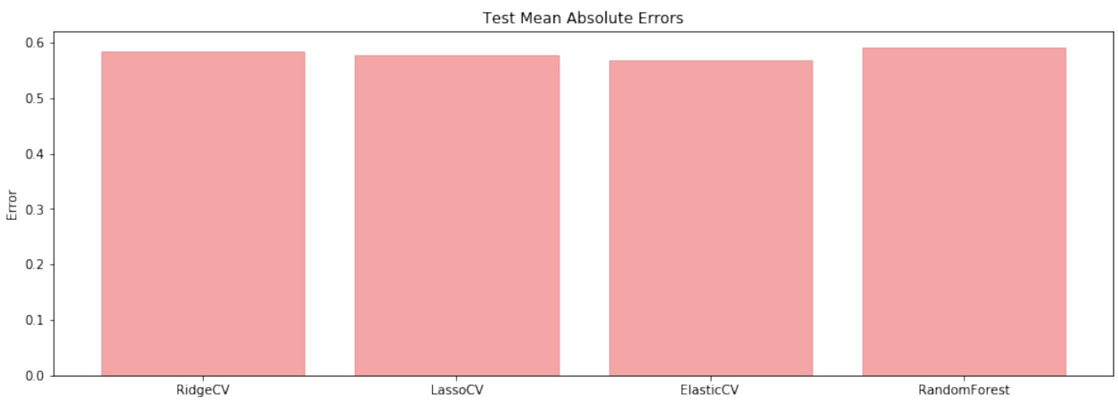
```
def compare_models(models):
    # Compute the training error for each model
    training_rmse = [rmse_score(model, X_train, Y_train) for model in models.
 ↪values()]
    # Compute the cross validation error for each model
    validation_rmse = [np.mean(cross_val_score(model, X_train, Y_train,␣
 ↪scoring=rmse_score, cv=5))
                       for model in models.values()]
    # Compute the test error for each model (don't do this!)
    test_rmse = [rmse_score(model, X_test, Y_test) for model in models.values()]
    names = list(models.keys())
    fig = go.Figure([
        go.Bar(x = names, y = training_rmse, name="Training RMSE"),
        go.Bar(x = names, y = validation_rmse, name="CV RMSE"),
        go.Bar(x = names, y = test_rmse, name="Test RMSE", opacity=.3)])
    fig.update_yaxes(title="RMSE")
    return fig
```

[36]:
```
compare_models(models)
```

As we can see, the linear models with regularization performed relativelty similar to each other for Training and CV RMSE, but the Ridge CV performed the best for Test RMSE. While the Random Forset had a slightly higher CV RMSE compared to the other models, it did much better in the Test RMSE. I would guess that since there is a much smaller training set to work with in cross validation, the model was not able to take advantage of its randomness, which works when there is a lot of data to work with. To validate this, let's also compare the test scores for these models with r2 score and mean average error. (EDIT: The second time I ran through this notebook, the Test RMSE of the Random Forest drastically increased. If the person grading this could let me know why the r2 score went from 0.8 the first time to around 0.2 the second time, it would be much appreciated, since I am curious about why there was such a large variance)

[37]:
```
from sklearn.metrics import mean_absolute_error

pred_ridge = ridge.predict(X_test)
pred_lasso = lasso.predict(X_test)
pred_elastic = elastic.predict(X_test)
pred_forest = forest.predict(X_test)

ridge_r2 = mean_absolute_error(Y_test,pred_ridge)
lasso_r2 = mean_absolute_error(Y_test,pred_lasso)
elastic_r2 = mean_absolute_error(Y_test,pred_elastic)
forest_r2 = mean_absolute_error(Y_test,pred_forest)

Models = ("RidgeCV", "LassoCV", "ElasticCV", "RandomForest")
scores = [ridge_r2, lasso_r2, elastic_r2, forest_r2]

plt.figure(1, figsize=(15, 5))
```

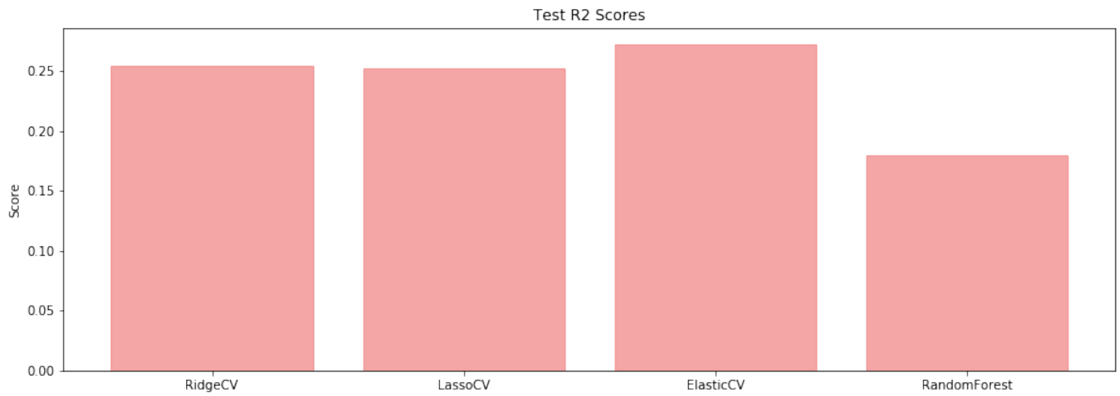
```
y_pos = np.arange(len(Models))
bargraph=plt.bar(y_pos, scores, alpha=0.7)
plt.xticks(y_pos, Models)
for i in range(len(Models)):
    bargraph[i].set_color('lightcoral')
plt.ylabel('Error')
plt.title('Test Mean Absolute Errors')
plt.show()
```



The Elastic model has a lower test Mean Absolute Error compared to the other models. Lets also compare test r2 scores.

[38]:
```
from sklearn.metrics import r2_score

pred_ridge = ridge.predict(X_test)
pred_lasso = lasso.predict(X_test)
pred_elastic = elastic.predict(X_test)
pred_forest = forest.predict(X_test)

ridge_r2 = r2_score(Y_test,pred_ridge)
lasso_r2 = r2_score(Y_test,pred_lasso)
elastic_r2 = r2_score(Y_test,pred_elastic)
forest_r2 = r2_score(Y_test,pred_forest)

Models = ("RidgeCV", "LassoCV", "ElasticCV", "RandomForest")
scores = [ridge_r2, lasso_r2, elastic_r2, forest_r2]

plt.figure(1, figsize=(15, 5))
y_pos = np.arange(len(Models))
bargraph=plt.bar(y_pos, scores, alpha=0.7)
plt.xticks(y_pos, Models)
for i in range(len(Models)):
    bargraph[i].set_color('lightcoral')
```

```
plt.ylabel('Score')
plt.title('Test R2 Scores')
plt.show()
```



As we can see, the Elastic model is much better than the other models with an r2 score of around 0.2. Based on all of this information, the best model to use for this data is the ElasticCV

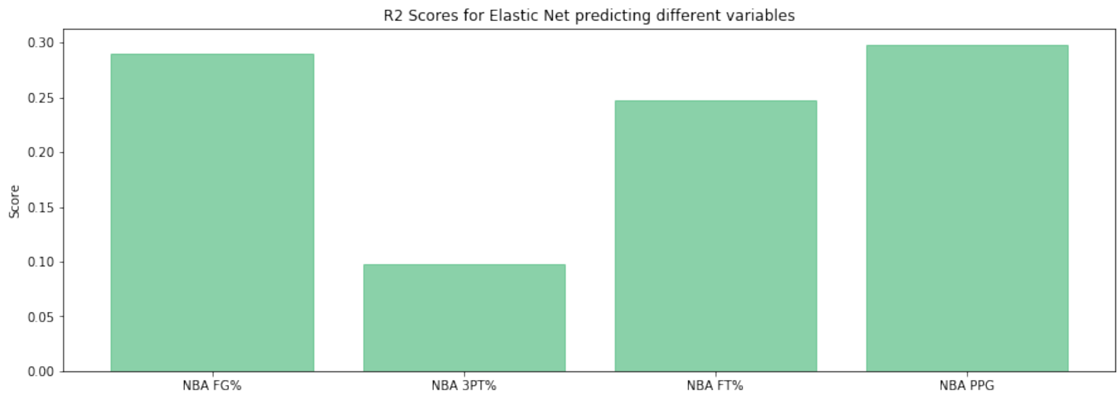
1.5 Interpretation/Summary

First, let's quickly see how well the ElasticCV predicts the other feautures we were interested in.

[39]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)

elastic.fit(X_train, Y_train["NBA_fg%"])
fg_pred = elastic.predict(X_test)
elastic.fit(X_train, Y_train["NBA__3ptpct"])
three_pred = elastic.predict(X_test)
elastic.fit(X_train, Y_train["NBA_ft%"])
ft_pred = elastic.predict(X_test)
elastic.fit(X_train, Y_train["NBA_ppg"])
ppg_pred = elastic.predict(X_test)
```

[40]:
```
fg_r2 = r2_score(Y_test["NBA_fg%"],fg_pred)
three_r2 = r2_score(Y_test["NBA__3ptpct"],three_pred)
ft_r2 = r2_score(Y_test["NBA_ft%"],ft_pred)
ppg_r2 = r2_score(Y_test["NBA_ppg"],ppg_pred)

Models = ("NBA FG%", "NBA 3PT%", "NBA FT%", "NBA PPG")
scores = [fg_r2, three_r2, ft_r2, ppg_r2]

plt.figure(1, figsize=(15, 5))
y_pos = np.arange(len(Models))
```

```
bargraph=plt.bar(y_pos, scores, alpha=0.6)
plt.xticks(y_pos, Models)
for i in range(len(Models)):
    bargraph[i].set_color('mediumseagreen')
plt.ylabel('Score')
plt.title('R2 Scores for Elastic Net predicting different variables')
plt.show()
```



As we can see from this graph, of all the features, the NCAA data is the worst at predicting the 3 Point percentage of players once they reach the NBA. It is decent at predicting Field Goal percentage, Free Throw percentage, and Points Per Game. The relatitvely low r2 score for all of them implies a weak correlation, however.

I was a little surprised by the accuracy of the model and the NCAA data in predicting NBA success, but to a degree it does make some sense. While NCAA performance can be indicative of general basketball skill, a multitude of other factors, including team, injuries, and mentality can play an important role in how successfull a player is in the NBA.

I felt that I correctly chose some models to use for my needs, and it is not surprisign that the models that incorporated regularization did much better than a pure linear model. I was surprised, however, with how poorly the random forest performed in cross validation and test error.

For future work, I would spend more time tuning my models to further reduce the cross validation error. Additionally, I would conduct Principal Component Analysis to determine which of the features played the biggest role in predicting NBA success