ST445_final_project

October 6, 2019

1 Project: Prediction of NBA 2018-2019 Season Champion

Table of Contents

2 Background

The National Basketball Association (NBA) consists of 30 professional teams located in major cities across the United States and Canada, and divides its season into three major parts: the Regular Season, the Playoffs and the Finals. The top 8 teams in the Reagular season from Western and Eastern respectively will be prompted to the playoffs, and the naissance of NBA Championship will rise in every June.

Apparently, the team with the best performance will win the Championship including offense and defense, and I believe these performance could be refelected in the stats. Therefore, I decide to

use the Support Vector Machine (SVM) to fit the past 30 years' Palyoffs and predict the Champion of the current season (2018-2019).

(We Are Basket, 2019)

3 Data

3.1 Variable Descriptions

The dataset is from Basketball Reference (Basketball-Reference.com, 2019), I used Jupyter notebook to do **web scarping** and the data is obtained from three tables, including Team Per Game Stats, Team Shooting, and Miscellaneous Stats, I then used the Pandas to merge these three tables (see the full code in the "data_scarping.ipynb").

The dataset was collected from 1988 to 2018, considering condition before 1988 was very different from the present, so the statistics before 1988 is omit.

Considering strength of teams are stratified during regular season, the top and the tail both have a huge gap, and a few teams also put rotten in order to get good draft next year, we might only use the data in the playoffs.

In addition, NBA started to collecting the shooting data since 2001, as a result, the team shooting before 2001 is unavailable.

Read the data and clean the data:

```
In [2]: import matplotlib.pyplot as plt
        %matplotlib inline
        import pandas as pd
        import numpy as np
        import warnings
        warnings.filterwarnings('ignore')

        # df.csv is gotten by the web scarping, and we clean the data
        df = pd.read_csv("df.csv").set_index("Team").drop("Unnamed: 0", axis=1)
        df_avg = df[df.index =='League Average'].set_index("Year") # df_avg only including dat
        df_nba = df[df.index !='League Average'].set_index("Year") #df_nba gets rid of data of
```

Import the color blending:

```
In [3]: import brewer2mpl
        bmap = brewer2mpl.get_map('Set2', 'qualitative', 8)
        colors = bmap.mpl_colors
```

3.2 Team Per Game Stats

```
In [4]: pd.set_option('display.max_columns', None) #set to display all columns
        print("The first table is about Tema Per Game Stats, detailed statistics description i
        df_avg.iloc[:, 0:21].head()
```

```
The first table is about Tema Per Game Stats, detailed statistics description in table:
```

Out[4]:

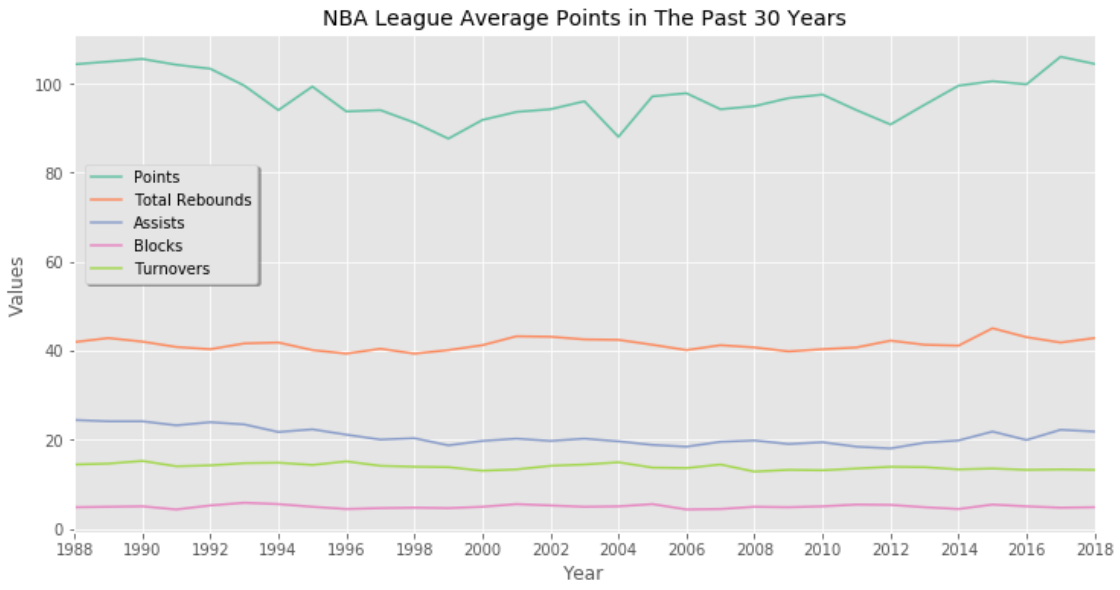
Year	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	FT	FTA
1988	40.2	84.3	0.478	1.8	5.8	0.314	38.4	78.5	0.490	22.0	28.4
1989	39.4	84.1	0.469	3.0	9.7	0.314	36.4	74.4	0.489	23.0	29.7
1990	39.9	84.0	0.475	2.6	8.1	0.316	37.3	75.9	0.492	23.2	30.5
1991	39.0	81.9	0.475	2.8	8.6	0.323	36.2	73.4	0.493	23.5	30.1
1992	38.8	81.7	0.475	2.8	8.3	0.337	36.0	73.4	0.490	22.9	29.7

Year	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
1988	0.777	13.8	28.1	41.9	24.4	7.2	4.8	14.4	23.7	104.3
1989	0.773	13.8	29.0	42.8	24.1	7.3	4.9	14.6	24.8	104.9
1990	0.761	13.3	28.7	42.0	24.1	7.8	5.0	15.2	24.9	105.5
1991	0.780	12.5	28.3	40.8	23.2	7.5	4.3	14.0	24.3	104.2
1992	0.769	13.2	27.1	40.3	23.9	7.7	5.2	14.2	24.2	103.3

3.2.1 The trend of five basic statistcs

As this dataset is a time series, we could use the line chart to test any obvious trend, I selected 5 basic statistics: as points and assists indicate team's offensive impact, and rebound and blocks are the important parts for team's defense; a championship team should be good at both offense and defense with as little turnovers as possible. Roughly, there is no obviously trend for these five statistics, the line of points shows a slightly fluctuation, while other four lines remain stable.

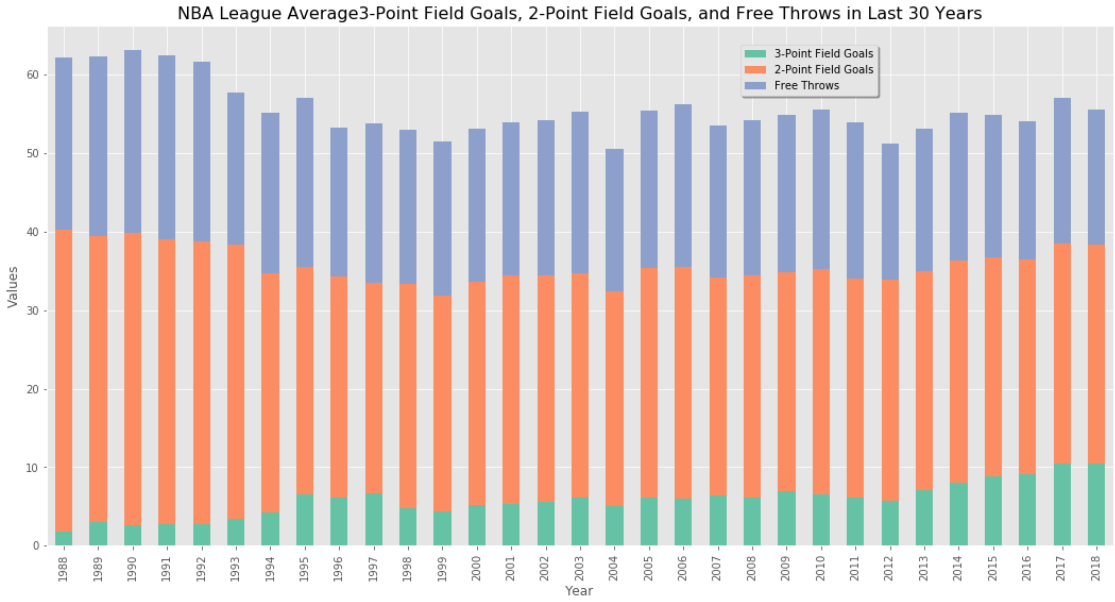
In [5]:
```
plt.style.use('ggplot') # choose the style as ggplot
df_avg[["PTS","TRB","AST","BLK","TOV"]].plot(figsize=(12,6),color = colors)
plt.xticks(np.arange(1988, 2020,2))
plt.xlim([1988,2018])
plt.legend(["Points", "Total Rebounds", "Assists", "Blocks", "Turnovers"], shadow = Tru
plt.ylabel("Values")
plt.title("NBA League Average Points in The Past 30 Years", fontsize = 14);
```



3.2.2 Changes in the way of scoring

Then points are made up of 3-points field goals, 2-points field goals and free throws, I uses stacked bar plot to see if the preferred way of goals had changed in last 30 years, as the below figure is shown, the percent of 3-points filed goals kept increasing, while this percent for 2-point field goals was decreasing, the free throws showed stable pattern. In conclusion, the NBA teams are more and more like to shoot 3 point, this result meets the NBA's "small ball era".

```
In [52]: df_avg[["3P","2P","FT"]].plot(kind = "bar", figsize=(18,9), stacked=True, color= colo
         plt.legend(["3-Point Field Goals", "2-Point Field Goals", "Free Throws"],loc=[0.65,0.8
         plt.ylabel("Values")
         plt.title("NBA League Average3-Point Field Goals, 2-Point Field Goals, and Free Throws
```



3.3 Team Shooting

The two tables below are about team shooting statistics from 2001 to 2018, the table not only includes field goal percent, but also includes the shooting distance from the baskets.

```
In [83]: df_avg.iloc[13:,22:35].head()
```

Out[83]:

Year	Dist.	2P(FGA)	0-3(FGA)	3-10(FGA)	10-16(FGA)	16 <3(FGA)	3P(FGA)	\
2001	12.2	0.815	0.263	0.164	0.185	0.203	0.185	
2002	12.2	0.792	0.293	0.150	0.137	0.212	0.208	
2003	12.4	0.785	0.288	0.146	0.135	0.216	0.215	

```
2004   12.2    0.797     0.297      0.135       0.139       0.226    0.203
2005   12.0    0.782     0.305      0.151       0.134       0.192    0.218

      2P(FG)  0-3(FG)  3-10(FG)  10-16(FG)  16 <3(FG)  3P(FG)
Year
2001   0.445    0.591     0.352      0.383      0.387   0.357
2002   0.456    0.593     0.380      0.366      0.378   0.336
2003   0.458    0.589     0.396      0.371      0.379   0.355
2004   0.445    0.576     0.362      0.376      0.366   0.324
2005   0.474    0.590     0.400      0.404      0.396   0.357
```

```
In [84]: df_avg.iloc[13:,35:45].head()
```

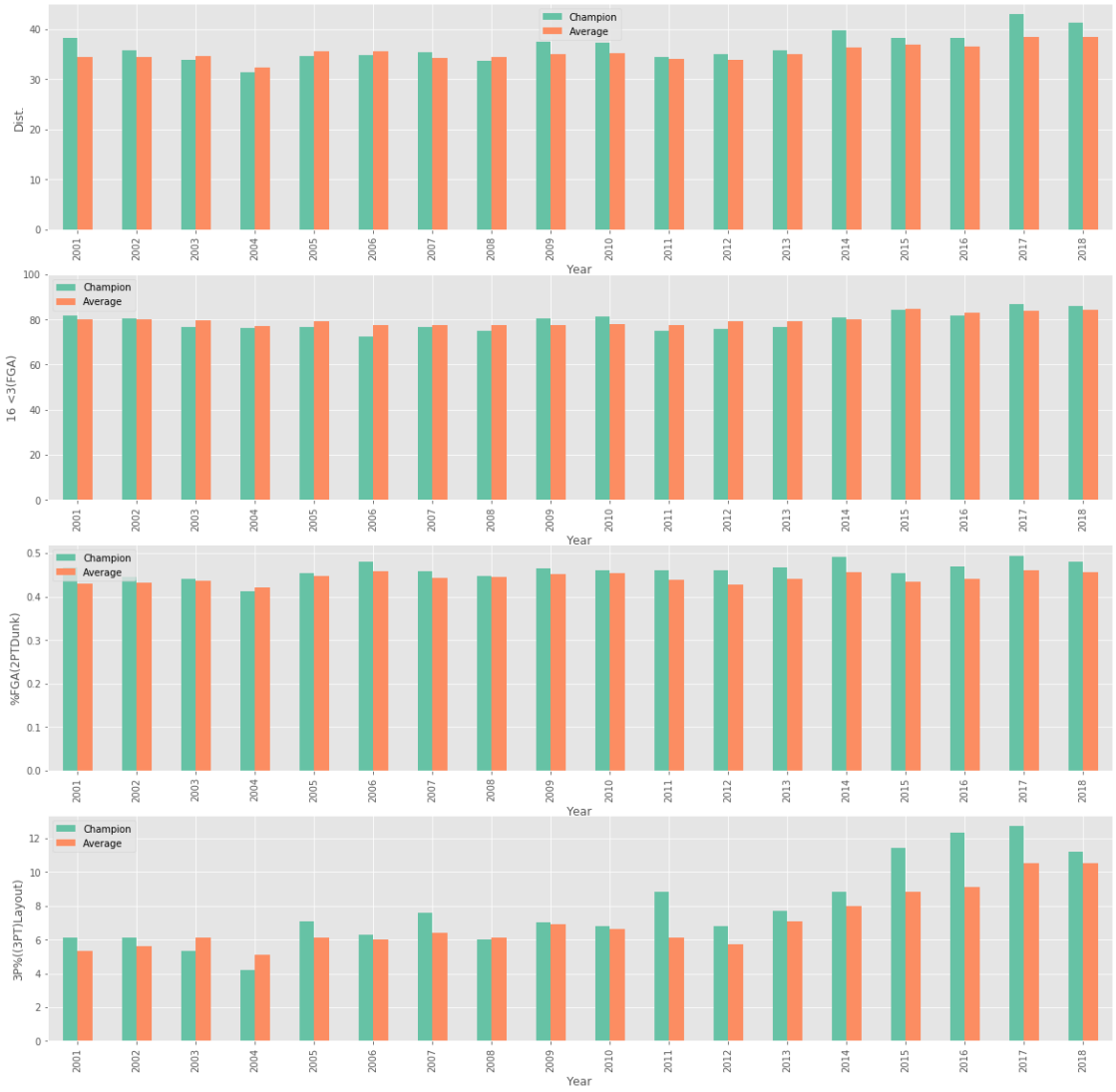
```
Out[84]:       %Ast'd(2PT)  %FGA(2PTDunk)  Md.(2PTDunk)  %FGA(2PTLayups)  \
Year
2001             0.542          0.039          23.0             0.000
2002             0.519          0.050          32.0             0.224
2003             0.534          0.043          34.0             0.226
2004             0.560          0.049          35.0             0.227
2005             0.467          0.056          42.0             0.233

      Md.(2PTLayups)  %Ast'd(3PT)  %3PA(3PTLayout)  3P%((3PT)Layout)  Att.  \
Year
2001             0.0        0.834            0.220             0.449   1.5
2002            84.0        0.841            0.245             0.364   0.8
2003           104.0        0.815            0.261             0.378   1.8
2004            90.0        0.857            0.259             0.345   1.6
2005            98.0        0.835            0.265             0.377   1.8

      Md.
Year
2001  0.0
2002  0.0
2003  0.0
2004  0.0
2005  0.1
```

3.3.1 Comparison of Champion and the average (in Bar Plot)

Similarly, there are no obviously trend in the team shooting, and also there werer small different btweem champion and the avereage; only the 3-point showed increasingly trend, and champion were more likely to try 3-points.

```
In [55]: stats = df_avg.columns[[22,27,36,42]]
         fig, axes = plt.subplots(nrows=4, ncols=1, figsize=(20,20))
         for i, stat in enumerate(stats):
             avg = pd.DataFrame(df_avg.iloc[13:,i])
             avg.columns = ["s"]
             compare = pd.DataFrame(df_nba[df_nba["Champion"]==1].iloc[13:,i]).join(avg)
```

```
        compare.columns= ["Champion", "Average"]
        compare.plot( color = colors[0:2], ax=axes[i], kind="bar")
        axes[i].set_ylabel(stat)
    axes[1].set_ylim(0,100);
```

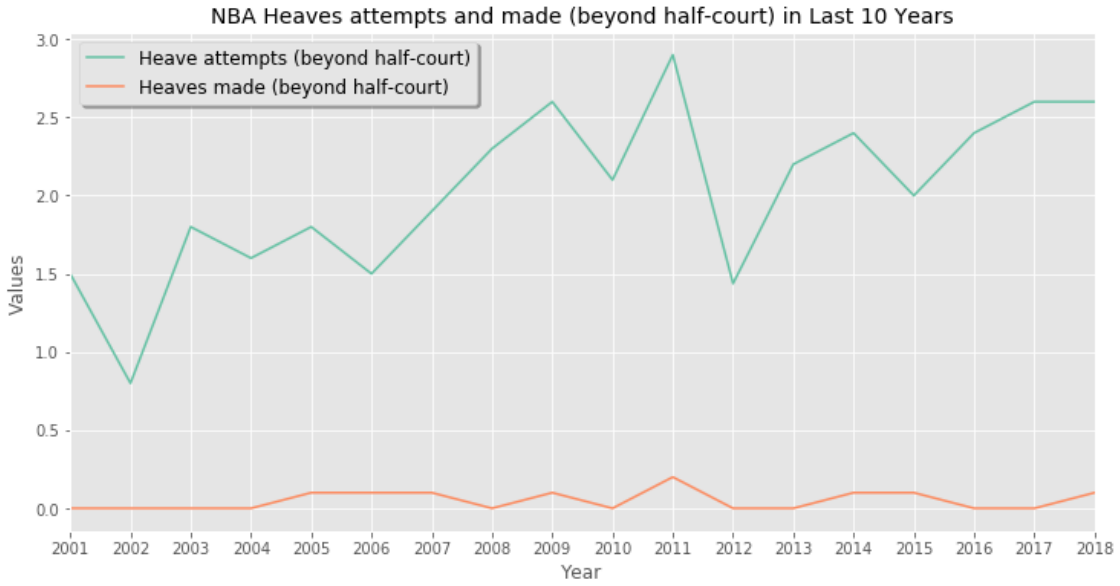


3.3.2 The Heaves Attempts

There is a significant increase in the heaves attempts (i.e. shooting beyond half-court), which may imply that NBA player are more encourage to try some lone-distance shooting attempt than before.

```
In [56]: df_avg.iloc[13:,43:45].plot(figsize=(12,6),color = colors[0:2])
         plt.xlim([2001,2018])
         plt.xticks(np.arange(2001, 2019,1))
```

```
plt.ylabel("Values")
plt.legend(["Heave attempts (beyond half-court)", "Heaves made (beyond half-court)"],
plt.title("NBA Heaves attempts and made (beyond half-court) in Last 10 Years", fontsi
```



3.3.3 Comparison of Champion and the average (in Radar Plot)

Intuitively, the performance of championship team should be better than other teams. For this season, I chose 2009,2014,2016,2017 (i.e. the championship in these years are very representative) and used the radar plot to compare the champion of these years' with the average level. Based on the figure below, the champion at least had one statistics which was much higher than the average. Despite 2016's Golden State Warrior, the champion stayed in the leading place in all aspects.

```
In [57]: import pandas as pd
         from math import pi
          # Set data
         years = [2009,2014,2016,2017]
         plt.figure(figsize=(12,14))
         for i,year in enumerate(years):
             #df_ra = df[df.index==year].iloc[[0,16]]
             df_ra = df[df["Year"] == year][df[df["Year"] == year]["Champion"] == 1].combine_f
             df_ra = df_ra[list(df.columns[30:36])]
             # ------- PART 1: Create background
             # number of variable
             categories=list(df_ra)[0:]
             N = len(categories)
             # What will be the angle of each axis in the plot? (we divide the plot / number o
             angles = [n / float(N) * 2 * pi for n in range(N)]
             angles += angles[:1]
```

```
# Initialise the spider plot
axes = plt.subplot(221+i, polar=True)
# If you want the first axis to be on top:
axes.set_theta_offset(pi / 2)
axes.set_theta_direction(-1)
#Draw one axe per variable + add labels labels yet
plt.xticks(angles[:-1], categories)
plt.tick_params(labelsize=12)
# Draw ylabels
axes.set_rlabel_position(0)
plt.yticks([0.25,0.5,0.75], ["0.25","0.5","0.75"], color="grey", size=7)
plt.ylim(0,0.69)
# ------- PART 2: Add plots
# Plot each individual = each line of the data
# I don't do a loop, because plotting more than 3 groups makes the chart unreadab
# Ind1
values=df_ra.iloc[0,].values.flatten().tolist()
values += values[:1]
axes.plot(angles, values, linewidth=1, linestyle='solid', label=df[df['Year']==yea
axes.fill(angles, values, 'b', alpha=0.1)
# Ind2
values=df_ra.iloc[1,].values.flatten().tolist()
values += values[:1]
axes.plot(angles, values, linewidth=1, linestyle='solid', label="League Average")
axes.fill(angles, values, 'r', alpha=0.1)
# Add legend
plt.legend(loc='best', bbox_to_anchor=(0.1, 0.1))
plt.title("NBA Champion's Field Goal% by Distance in " + str(year) +"\n", fontsiz
```

3.4 Miscellaneous Stats

This table is about some advanced statistics, which Revealing some potential winning factors. Specific Explaination for those statistixs is: **PW**: Pythagorean wins, i.e., expected wins based on points scored and allowed; **PL**: Pythagorean losses, i.e., expected losses based on points scored and allowed; **ORtg**: An estimate of points produced (players) or scored (teams) per 100 possessions; **DRtg**: An estimate of points allowed per 100 possessions; **Pace**: An estimate of possessions per 48 minutes; **FTr**: Number of FT Attempts Per FG Attempt; **3PAr**: Percentage of FG Attempts from 3-Point Range; **eFG%**: This statistic adjusts for the fact that a 3-point field goal is worth one more point than a 2-point field goal; **TOV%**: An estimate of turnovers committed per 100 plays; **ORB%**: An estimate of the percentage of available offensive rebounds a player grabbed while he was on the floor; **FT/FGA**: Free Throws Per Field Goal Attempt. "offens" means for offense factors, and "defense" means for opponent.

```
In [58]: print("Viewing the thrid table:")
         df_avg.iloc[:,45:-1].head()

Viewing the thrid table:
```

Out[58]:

Year	PW	PL	ORtg	DRtg	Pace	FTr	3PAr	eFG%(offens)	TOV%(offens)	\
1988	5.0	5.0	110.4	110.4	94.0	0.337	0.068	0.488	13.0	
1989	3.9	3.9	110.2	110.2	94.4	0.353	0.115	0.487	13.0	
1990	4.5	4.5	109.4	109.4	95.9	0.364	0.097	0.490	13.5	
1991	4.2	4.3	110.9	110.9	93.3	0.367	0.105	0.492	12.8	
1992	4.5	4.6	111.4	111.4	91.7	0.364	0.101	0.492	13.0	

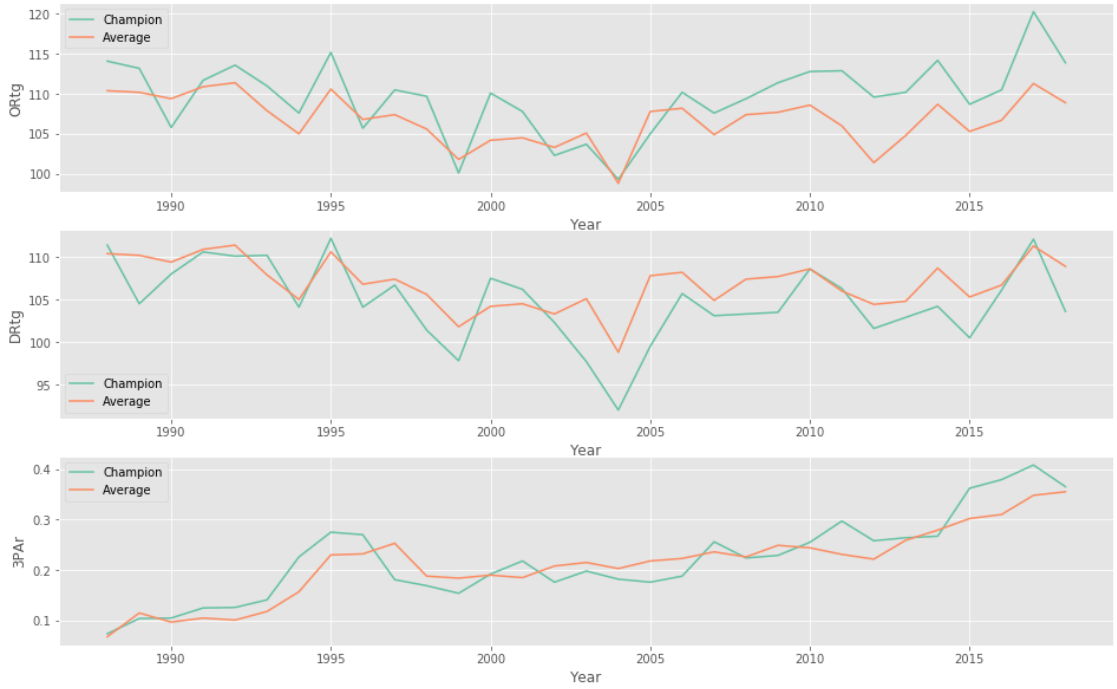
Year	ORB%(offens)	FT/FGA(offens)	eFG%(defense)	TOV%(defense)	\
1988	32.9	0.262	0.488	13.0	
1989	32.3	0.273	0.487	13.0	
1990	31.7	0.277	0.490	13.5	
1991	30.6	0.286	0.492	12.8	
1992	32.8	0.280	0.492	13.0	

Year	DRB%(defense)	FT/FGA(defense)
1988	67.1	0.262
1989	67.7	0.273
1990	68.3	0.277
1991	69.4	0.286
1992	67.2	0.280

3.4.1 Comparison of Champion and the average in ORtg, DRtg and 3PAr

Clearly, the champion teams would have higher ORtg and lower DRtg, and also Champions in recent years attempted more 3 points than the average, which may imply that those teams which adapted to "small ball era" are more likely to win the championship.

```
In [59]: stats = df_avg.columns[[47,48,51]]
         fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(16,8))
         for i, stat in enumerate(stats):
             avg = pd.DataFrame(df_avg[stat])
             avg.columns = ["s"]
             compare = pd.DataFrame(df_nba[df_nba["Champion"]==1][stat]).join(avg)
             compare.columns= ["Champion", "Average"]
             compare.plot(figsize=(16,10), color = colors[0:2], ax=axes[i])
             axes[i].set_ylabel(stat)
```

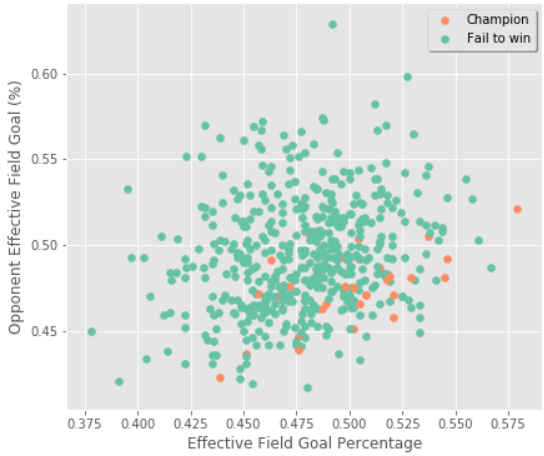


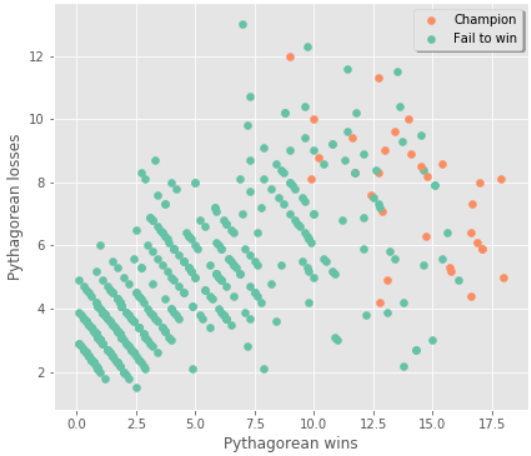
3.4.2 Defense Versus Offense and the Public Opinion

Each point from the below figures denotes a team, and those orange points represent the champion team. Opponent Effective Field Goal measures a team's defense, and the Effective Filed Goal Percentage measures the team's offense. The figure on the left shows that defense and offense are equally important. A few champions may have weak defense or offense, but they must have super offense or defense to make up their weakness; Pythagorean wins or losses show the expectation of the public. The figure on the right is showing nearly no dark house during the NBA off season.

```
In [60]: plt.figure(figsize=(15,6))
         plt.subplot(121)
         for i in range(0, len(df_nba)):
             plt.scatter(list(df_nba["eFG%(offens)"])[i], list(df_nba["eFG%(defense)"])[i], co
         plt.legend(['Champion','Fail to win'], shadow =True)
         plt.xlabel("Effective Field Goal Percentage")
         plt.ylabel("Opponent Effective Field Goal (%)")
         plt.subplot(122)

         print(" expected wins/losses based on points scored and allowed")
         for i in range(0, len(df_nba)):
             plt.scatter(list(df_nba["PW"])[i], list(df_nba["PL"])[i], color =colors[int(list(
         plt.legend(['Champion','Fail to win'], shadow = True)
         plt.xlabel("Pythagorean wins")
         plt.ylabel("Pythagorean losses");
```

expected wins/losses based on points scored and allowed

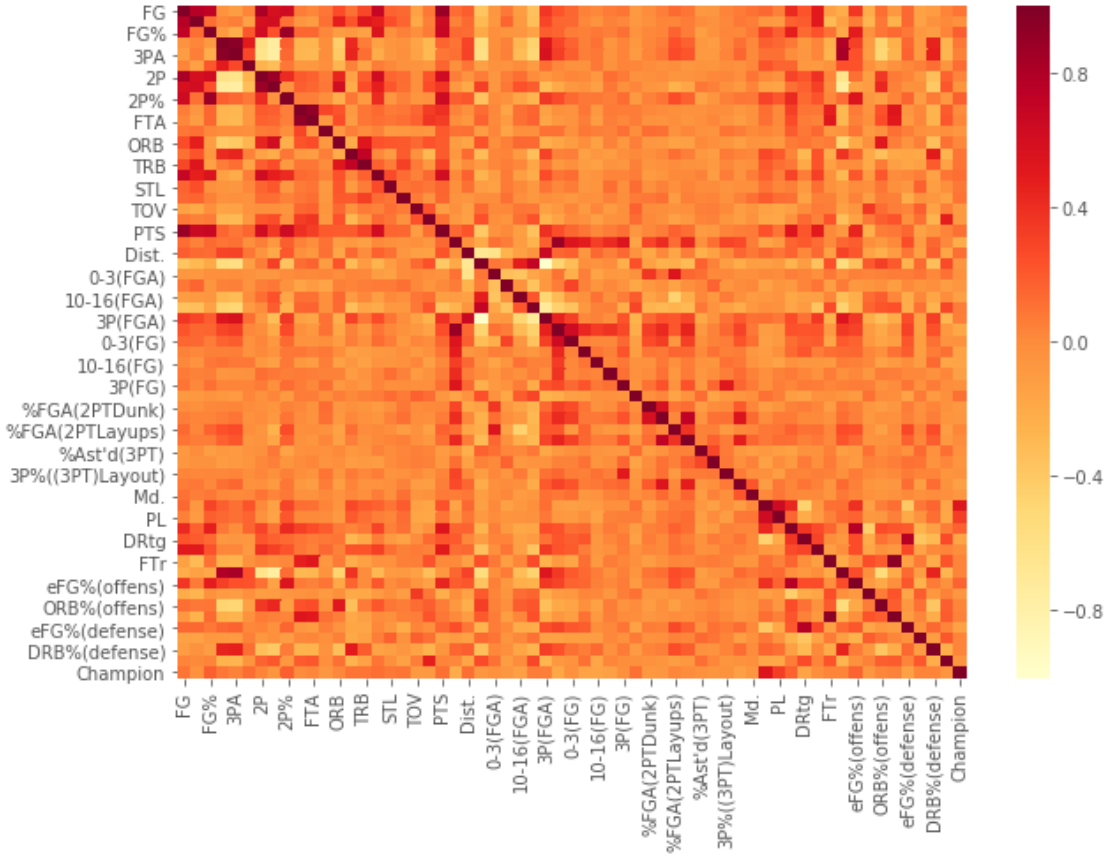




4 Analysis

4.1 Correlation Plot

```
In [7]: df_corr = df.reset_index()
        corrr = df_corr.drop("Team", axis = 1).drop("Year", axis = 1).corr()
        import seaborn as sns
        plt.figure(figsize=(10,7))
        ax = sns.heatmap(corrr, cmap=plt.cm.YlOrRd)
```




```
In [8]: print("The table shows the variable whose correlation was higher than 0.2")
        corrr[-1:][abs(corrr[-1:])>0.2].dropna(axis=1)
```

```
The table shows the variable whose correlation was higher than 0.2
```

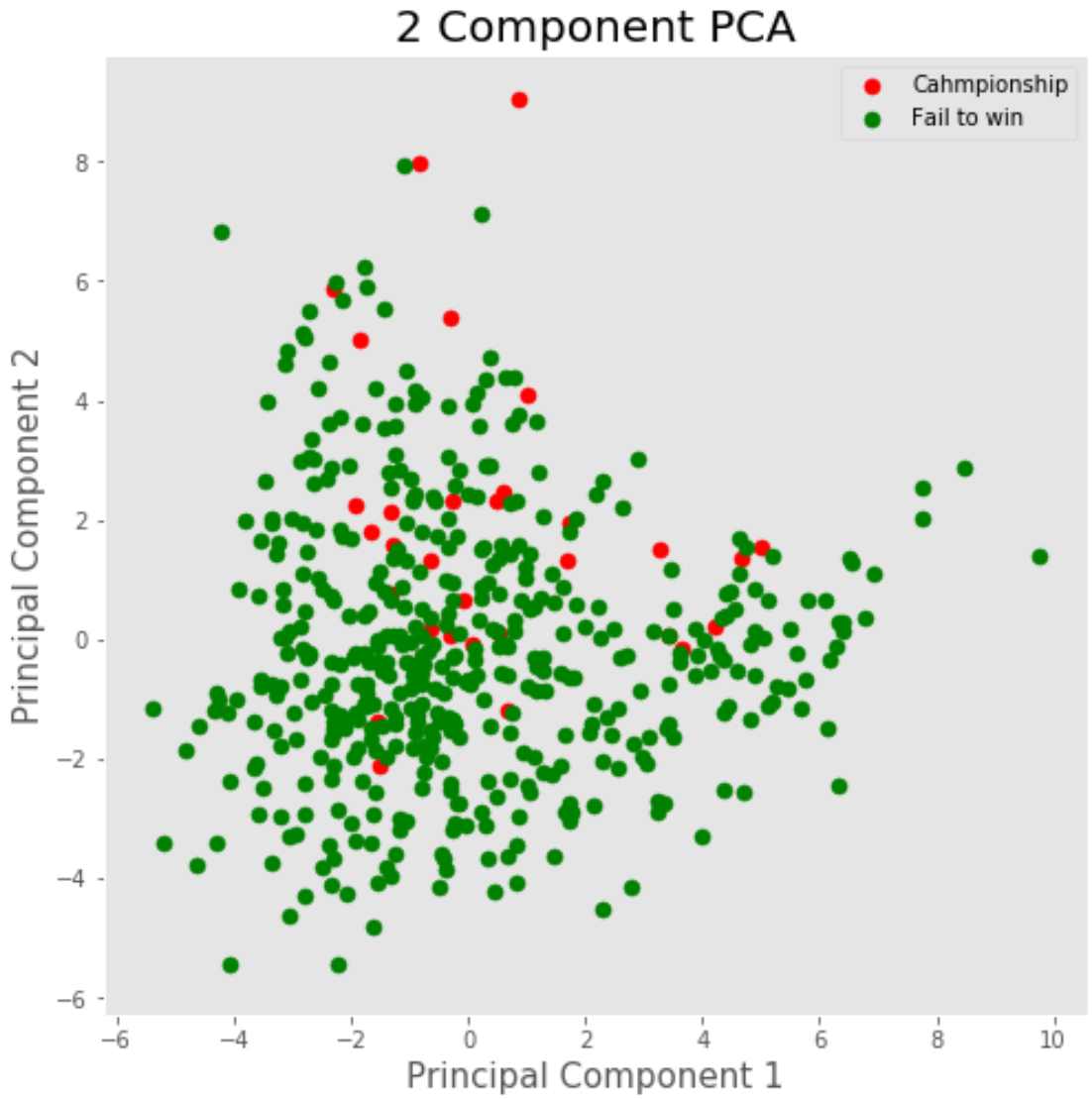
```
Out[8]:              FG%.1        PW        PL  Champion
        Champion -0.201011  0.537517  0.332717       1.0
```

4.2 PCA

As there are so many variables, so before fiitting the mdoel, it might be a good idea to reduce the dimensions by PCA, however, a poor result is shown when two groups were not separable.

```
In [63]: df_clean = df_nba.drop(list(df_nba)[21:45], axis=1)
         feature = list(df_clean)
         feature.remove('Champion')

         from sklearn.preprocessing import StandardScaler
         # Separating out the features
```

```
x = df_clean.loc[:, feature].values
# Separating out the target
y = df_clean.loc[:,['Champion']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
target = df_clean[['Champion']]
target = target.reset_index().drop('Year', axis =1)
finalDf = principalDf.join(target)
#Plot of PCA
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 Component PCA', fontsize = 20)
targets = [1, 0]
colors = ['r', 'g']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Champion'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(['Cahmpionship', 'Fail to win'])
ax.grid()
```



4.3 Support Vector Machine (SVM)

Dealing with the missing in the team shooting table:

```
In [64]: from sklearn.preprocessing import Imputer
         # missing_values is the value of your placeholder, strategy is if you'd like mean, me
         imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
         imp.fit(df_nba)
         Imputer(axis=0, copy=True, missing_values='NaN', strategy='mean', verbose=0)
         df_arr = imp.transform(df_nba)
```

Randomly spilting the training and test data:

```
In [76]: import random
         random.seed(8)
         df_clean = df_nba.drop(list(df_nba)[21:45], axis=1)
         feature = list(df_clean)
         feature.remove('Champion')
         from sklearn.preprocessing import StandardScaler
         # Separating out the features
         X = df_clean.loc[:, feature].values
         X = StandardScaler().fit_transform(X)
         # Separating out the target
         y = np.array(df_clean)[:,-1]
         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

Applying the SVM and testing the performance, display the classification report:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2TP}{2TP + FP + FN}$$

```
In [77]: from sklearn.svm import SVC
         clf = SVC(kernel='linear', class_weight={1: 15})
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         from sklearn.metrics import classification_report, confusion_matrix
         print(confusion_matrix(y_test, y_pred))
         print(classification_report(y_test, y_pred))
```

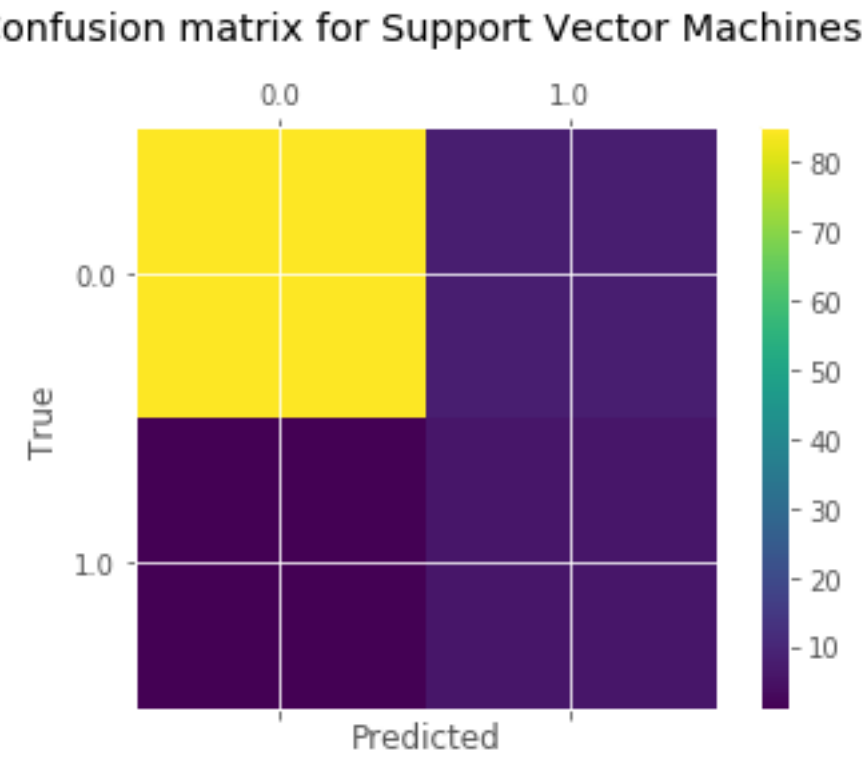
```
[[85  8]
 [ 1  6]]
             precision    recall  f1-score   support

        0.0       0.99      0.91      0.95        93
        1.0       0.43      0.86      0.57         7

avg / total       0.95      0.91      0.92       100
```

4.3.1 Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing (Google.com, 2018).

In [78]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

def draw_confusion_matrices(confusion_matricies,class_names):
    class_names = class_names.tolist()
    for cm in confusion_matrices:
        classifier, cm = cm[0], cm[1]
        print(cm)

        fig = plt.figure()
        ax = fig.add_subplot(111)
        cax = ax.matshow(cm)
        plt.title('Confusion matrix for Support Vector Machines \n')
        fig.colorbar(cax)
        ax.set_xticklabels([''] + class_names)
        ax.set_yticklabels([''] + class_names)
        plt.xlabel('Predicted')
        plt.ylabel('True')
        plt.show()

confusion_matrices = [( "Support Vector Machines", confusion_matrix(y_test, y_pred) )]
draw_confusion_matrices(confusion_matrices,np.unique(y))
```

```
[[85  8]
 [ 1  6]]
```



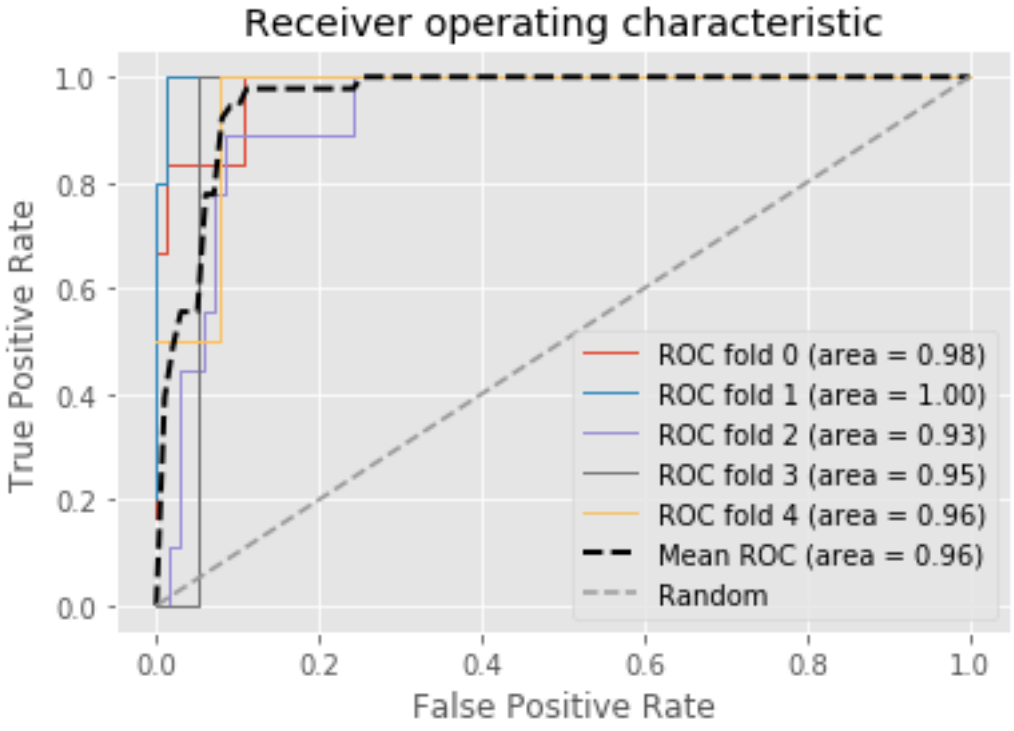
4.3.2 ROC Curve

A receiver operating characteristic curve, (i.e., ROC curve), is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied (En.wikipedia.org, 2018). The ROC curve shows that the model was fitted very well.

```
In [79]: from sklearn.metrics import roc_curve, auc
         from scipy import interp
         from sklearn.cross_validation import KFold
         from sklearn.svm import SVC

         def plot_roc(X, y, clf_class, **kwargs):
             kf = KFold(len(y), n_folds=5, shuffle=True)
             y_prob = np.zeros((len(y),2))
             mean_tpr = 0.0
             mean_fpr = np.linspace(0, 1, 100)
             all_tpr = []
             for i, (train_index, test_index) in enumerate(kf):
                 X_train, X_test = X[train_index], X[test_index]
                 y_train = y[train_index]
                 clf = clf_class(**kwargs)
                 clf.fit(X_train,y_train)
                 # Predict probabilities, not classes
```

```
            y_prob[test_index] = clf.predict_proba(X_test)
            fpr, tpr, thresholds = roc_curve(y[test_index], y_prob[test_index, 1])
            mean_tpr += interp(mean_fpr, fpr, tpr)
            mean_tpr[0] = 0.0
            roc_auc = auc(fpr, tpr)
            plt.plot(fpr, tpr, lw=1, label='ROC fold %d (area = %0.2f)' % (i, roc_auc))
        mean_tpr /= len(kf)
        mean_tpr[-1] = 1.0
        mean_auc = auc(mean_fpr, mean_tpr)
        plt.plot(mean_fpr, mean_tpr, 'k--',label='Mean ROC (area = %0.2f)' % mean_auc, lw=

        plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Random')
        plt.xlim([-0.05, 1.05])
        plt.ylim([-0.05, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic')
        plt.legend(loc="lower right")
        plt.show()
    print("The ROC Curve Plot:")
    plot_roc(X_train,y_train,SVC, probability=True)
```

```
The ROC Curve Plot:
```



4.4 Prediction

I used the current season's stats (collected at 18/12/2018) to predict the championship of this season:

```
In [80]: # import current season's data
         current = pd.read_csv("current.csv")
         pred = current.drop(list(current)[21:45], axis=1).drop("Unnamed: 0", axis =1).drop("T
         result = clf.predict(StandardScaler().fit_transform(pred))
         print("Only one team were predicted as '1',  which are:")
         pd.DataFrame(current["Team"])[result == 1]

Only one team were predicted as '1',  which are:


Out[80]:              Team
         0  Toronto Raptors
```

The prediction seems reasonable, Toronto Raptors is now at the top place in the NBA, they indeed perform best at present. According to Williamhill, Toronto Raptors has the second highest odd (Sports.williamhill.com, 2018), the highest odd is Golden State Warrior, because my cannot model can't explain the effect of temporary injuries in Warrior. But anyway, it seems that Raptors has a great grasp to the finals, and who is going to win the championship? We will see...

5 Reference

Basketball-Reference.com. (2019). Basketball Statistics and History | Basketball-Reference.com. [online] Available at: https://www.basketball-reference.com [Accessed 12 Jan. 2019]. En.wikipedia.org. (2019). Receiver operating characteristic. [online] Available at: https://en.wikipedia.org/wiki/Receiver_operating_characteristic [Accessed 12 Jan. 2019]. Sports.williamhill.com. (2019). [online] Available at: http://sports.williamhill.com/bet/en-gb/betting/g/630/Championship+Winner.html [Accessed 12 Jan. 2019]. We Are Basket. (2019). Top 5 Most Championships — We Are Basket. [online] Available at: http://wearebasket.net/top-5-championships/ [Accessed 12 Jan. 2019].