Betting Odds in Soccer: An Analysis on Home Team Advantage and Individual Player Skill

Richard Chen, Keshav Sota, Angela Zhou
Data Bootcamp
Spring 2018

Abstract

Betting agencies have long existed in the sporting world and are often seen as a form of gambling dictated by team loyalty. We dive deeper into potential factors that influence betting odds. Our analysis leads us to the conclusion that a Home Team Advantage depresses Home Betting Odds across all leagues and that individual star players may also have an effect, although a concrete conclusion is more difficult due to confounding variables across leagues.

Data Input and Libraries

The data used comes from preset data downloaded from Kaggle. Provided is data from 25,000+ matches, 10,000+ players, 11 European Countries, and betting odds from multiple providers spanning 2008 and 2016. Player and Team data has been sourced from the EA Fifa Video Game Francise. We used Bet 365 Data for our Betting Odds. Data was imported into the Jupyter Notebook using the following code:

In [2]:
```
import sys
import numpy as np #analysis
import pandas as pd
import matplotlib.pyplot as plt #plotting vizualizations
from math import pi
import datetime
import seaborn as sns #statistical graphs
from scipy.stats.stats import linregress

plt.style.use("ggplot") #data vizualization

%matplotlib inline
```

In [3]:
```
path = "C://Users//Ricky//Desktop//DB Final//Player_Attributes.csv"
path2 = "C://Users//Ricky//Desktop//DB Final///Player.csv"
path3 = "C://Users//Ricky//Desktop//DB Final//Match.csv"
path4 = "C://Users//Ricky//Desktop//DB Final//Team.csv"
PlayerData1 = pd.read_csv(path)
PlayerNames1 = pd.read_csv(path2)
MatchData1 = pd.read_csv(path3)
TeamData1 = pd.read_csv(path4)

#print("Variables dtypes:\n:",PlayerData1.dtypes, sep='')
#print("Variables dtypes:\n:",PlayerNames1.dtypes, sep='')
```

This section of the code links together a few files. In this instance we are linking the Player Name/Player ID key on one CSV file to the Player Attribute CSV file, which only contains Player ID information. We similarly do this on later with Team Names on the Match CSV file and the Team CSV file to link all the information together.

In [4]:
```
PlayerData = PlayerData1[['player_fifa_api_id', 'player_api_id','overall_rating', 'preferred_foot', 'attacking_work_rate', 'defensive_work_rate', 'crossing', 'finishing', 'heading_accuracy', 'short_passing', 'volleys', 'dribbling', 'curve', 'free_kick_accuracy', 'long_passing', 'ball_control', 'acceleration', 'sprint_speed', 'agility', 'reactions', 'balance', 'shot_power', 'jumping', 'stamina', 'strength', 'long_shots', 'aggression', 'interceptions', 'positioning', 'vision', 'penalties', 'marking', 'standing_tackle', 'sliding_tackle', 'gk_diving', 'gk_handling', 'gk_kicking', 'gk_positioning', 'gk_reflexes']]
# Player attributes were called from the data and used to characterize each player
PlayerNames = PlayerNames1[["player_api_id","player_name"]]
# This would later be used to match player API ID's to Player Names
TopPlayers = PlayerData[PlayerData.overall_rating >= 80]
# This was used to select top players
MatchData = MatchData1[["id","league_id","season","match_api_id","home_team_api_id","away_team_api_id","home_team_goal", "away_team_goal", "home_player_1", "home_player_2","home_player_3",'home_player_4','home_player_5','home_player_6','home_player_7','home_player_8','home_player_9','home_player_10','home_player_11','away_player_1','away_player_2','away_player_3','away_player_4','away_player_5','away_player_6','away_player_7','away_player_8','away_player_9','away_player_10','away_player_11','B365H','B365D','B365A']]
# Key match variables were called from the data and used in analysis
TeamData = TeamData1[["team_api_id","team_long_name","team_short_name"]]
# This will later be used to match team API ID's to Team Names
```

The first step in our analysis was to take a look at matches from a high-level. In order to do this, we began by matching MatchData team API ID values to their respective team names.

In [5]:
```
HomeTeamData = TeamData.rename(columns={'team_api_id':'home_team_api_id', 'team_long_name':'home_team_long_name','team_short_name':'home_team_short_name'})
AwayTeamData = HomeTeamData.rename(columns={'home_team_api_id':'away_team_api_id', 'home_team_long_name':'away_team_long_name','home_team_short_name':'away_team_short_name'})
#These two dataframes were made to facilitate a simple merge.
```

In [6]:
```
print('Dimensions of MatchData:', MatchData.shape)
print('Dimensions of HomeTeamData:', HomeTeamData.shape)

HalfMatchData = pd.merge(MatchData, HomeTeamData,   #First we merged home
 team IDs
                 how='left',
                 on='home_team_api_id',
                 indicator=False) #Since multiple merges will be necessar
y, we set this False for now.

print('Dimensions of new df:', HalfMatchData.shape)
```

```
Dimensions of MatchData: (25979, 33)
Dimensions of HomeTeamData: (299, 3)
Dimensions of new df: (25979, 35)
```

In [7]:
```
print('Dimensions of HalfMatchData:', HalfMatchData.shape)
print('Dimensions of AwayTeamData:', AwayTeamData.shape)

AllMatchData = pd.merge(HalfMatchData, AwayTeamData,   #The new dataframe will
 have both away and home team names matched.
                 how='left',
                 on='away_team_api_id',
                 indicator=True) #We now set this True to remember a merge has
 occured.

print('Dimensions of new df:', AllMatchData.shape)
```

```
Dimensions of HalfMatchData: (25979, 35)
Dimensions of AwayTeamData: (299, 3)
Dimensions of new df: (25979, 38)
```

Now that we have merged these two dataframes, we are ready to begin the first step of our analysis.

Investigating Home Team Advantage

In [8]:
```
HomeTeamAdv=AllMatchData[["home_team_long_name", "home_team_goal", "away_team_
long_name", "away_team_goal"]]
#The following dataframe contains very basic statistics about each match.
```

In [9]:
```
HomeTeamAdv["net_goal"]=HomeTeamAdv["home_team_goal"]-HomeTeamAdv["away_team_g
oal"]
HomeTeamAdv.head(10)
#Here, we add a new column that will help facilitate our analysis. It marks th
e difference in goals between the two teams.
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Out[9]:

	home_team_long_name	home_team_goal	away_team_long_name	away_team_goal
0	KRC Genk	1	Beerschot AC	1
1	SV Zulte-Waregem	0	Sporting Lokeren	0
2	KSV Cercle Brugge	0	RSC Anderlecht	3
3	KAA Gent	5	RAEC Mons	0
4	FCV Dender EH	1	Standard de Liège	3
5	KV Mechelen	1	Club Brugge KV	1
6	KSV Roeselare	2	KV Kortrijk	2
7	Tubize	1	Royal Excel Mouscron	2
8	KVC Westerlo	1	Sporting Charleroi	0
9	Club Brugge KV	4	KV Kortrijk	1

In [10]:
```
HomeTeamAdv.shape
```

Out[10]:
```
(25979, 5)
```

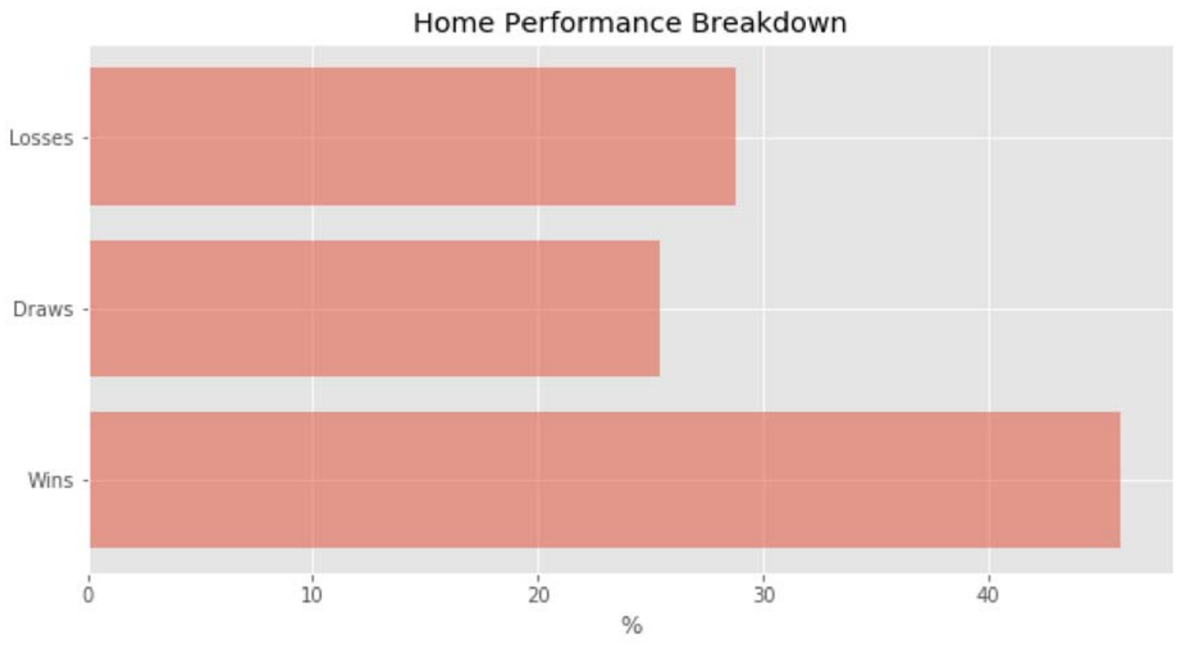
In [11]:
```
number_games = len(HomeTeamAdv)
number_wins = sum(a>0 for a in HomeTeamAdv["net_goal"])
number_draw = sum(a==0 for a in HomeTeamAdv["net_goal"])
number_loss = sum(a<0 for a in HomeTeamAdv["net_goal"])
print("Total Games:", number_games)
print("Total Home Wins:", number_wins)
print ("Total Home Draws:", number_draw)
print ("Total Home Loss:", number_loss)
#Using this code, we are able to identify the total number of games won, draw
n, and lost for the Home Team.
```

```
Total Games: 25979
Total Home Wins: 11917
Total Home Draws: 6596
Total Home Loss: 7466
```

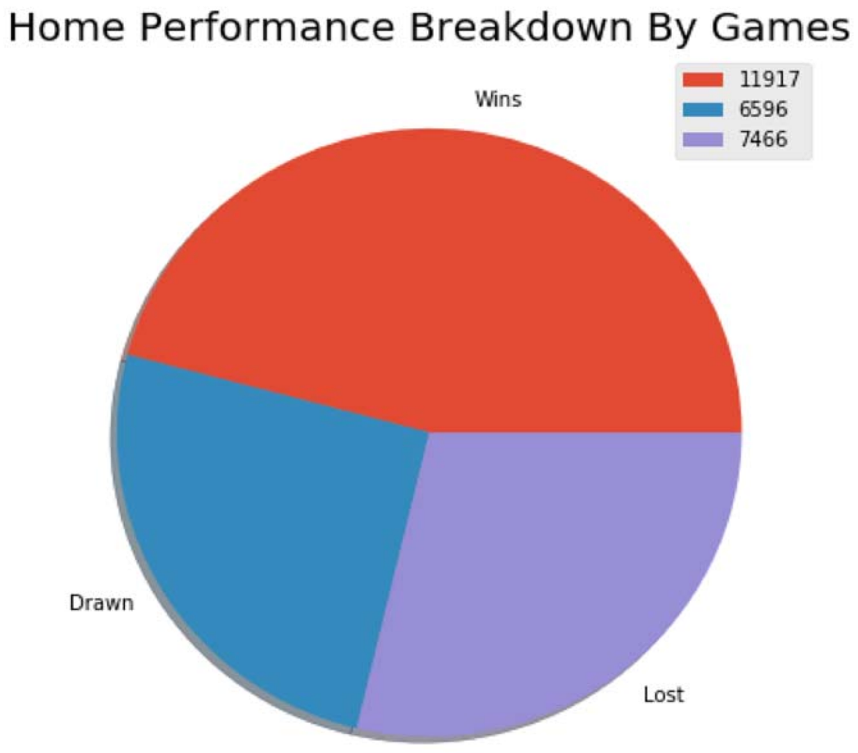
In [12]:
```
percentage_wins = number_wins/number_games*100
print(percentage_wins,"% Games Won")
percentage_draw = number_draw/number_games*100
print(percentage_draw,"% Games Won")
percentage_loss = number_loss/number_games*100
print(percentage_loss, "% Games Won")
#We also use the following code to look at the percentage of games that respec
tively won, drawn, and lost for the Home Team.
```

```
45.87166557604219 % Games Won
25.389737865198814 % Games Won
28.738596558759 % Games Won
```

In [13]:
```
plt.figure(figsize=(10,5))
percentages = [percentage_wins, percentage_draw, percentage_loss]
labels = ["Wins","Draws","Losses"]
y_pos = np.arange(len(percentages))
plt.barh(y_pos, percentages, align='center', alpha=0.5)
plt.yticks(y_pos, labels)
plt.xlabel(r'%')
plt.title('Home Performance Breakdown')
plt.show()

#This simple horizontal bar chart allows us to visualize the difference in pro
portions of games won, drawn, and lost.
```



In [14]:
```
plt.figure(figsize=(7,7))
breakdown = [number_wins, number_draw, number_loss]
plt.pie(breakdown, labels=["Wins", "Drawn", "Lost"], shadow=True)
plt.title('Home Performance Breakdown By Games', fontsize=20)
plt.legend([number_wins, number_draw, number_loss])
plt.show()

#An accompagnying pie chart is also included for further visualization.
```



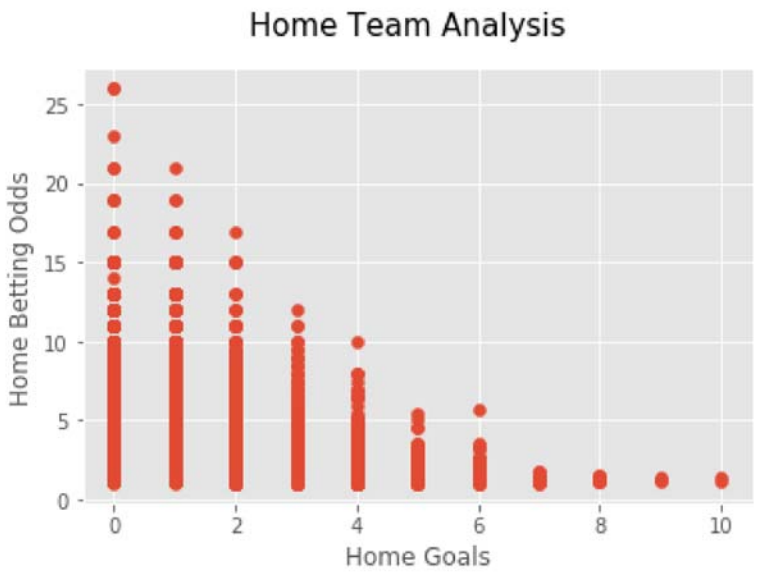
Given the overwhelming amount of Home wins compard to losses and draws, we believed it reasonable to conclude that a Home Advantage could exist. We can now investigate whether home team advantage has an effect on betting odds on a match by match basis..

In [15]:
```
MatchAnalysis=AllMatchData[["home_team_long_name", "home_team_goal","away_team
_long_name","away_team_goal","B365H","B365D","B365A"]]
MatchAnalysis.head()
#We create a new dataframe to eliminate data that is not needed for the analys
is
#We let the number of home goals be a proxy for a combination of home field ad
vantage and relative skill in the match.
#We will run a regression on home_team_goal and B365H to examine whether an ap
parent correlation exists.
```

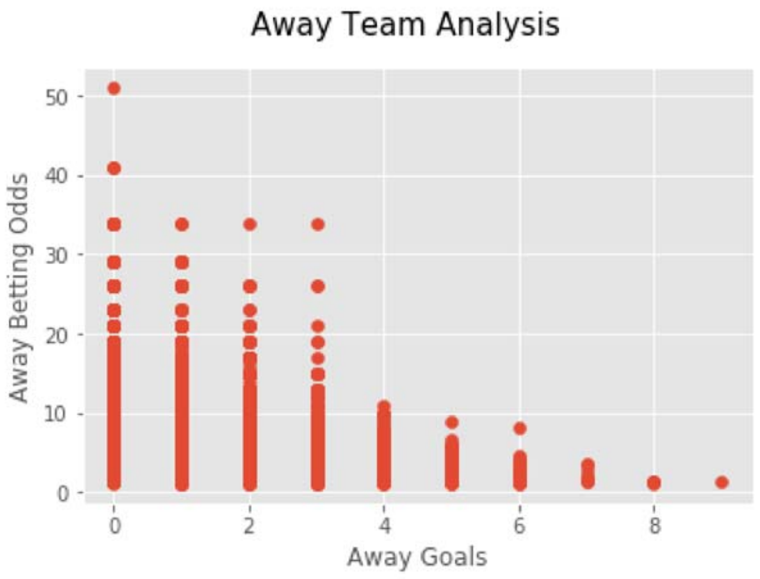
Out[15]:

	home_team_long_name	home_team_goal	away_team_long_name	away_team_goal
0	KRC Genk	1	Beerschot AC	1
1	SV Zulte-Waregem	0	Sporting Lokeren	0
2	KSV Cercle Brugge	0	RSC Anderlecht	3
3	KAA Gent	5	RAEC Mons	0
4	FCV Dender EH	1	Standard de Liège	3

In [18]:
```
plt.scatter(MatchAnalysis["home_team_goal"],MatchAnalysis["B365H"])
plt.xlabel("Home Goals")
plt.ylabel("Home Betting Odds")
plt.suptitle("Home Team Analysis", fontsize = 15)
plt.show()
#Our analysis of this scatterplot should show a negative relationship between
 goals or "relative strength" and home betting odds
#We will examine the same for away statistics, but expect a larger spread in b
etting values due to a lack of home team advantage
```



In [19]:
```
plt.scatter(MatchAnalysis["away_team_goal"],MatchAnalysis["B365A"])
plt.xlabel("Away Goals")
plt.ylabel("Away Betting Odds")
plt.suptitle("Away Team Analysis", fontsize = 15)
plt.show()
```



The difference between the two scatterplots is slight but significant. The data maps as we would expect. On a match by match basis, we see that home team betting odds are more heavily concentrated towards par 1:1 while away team betting odds experience a much larger spread. We keep in mind that goals scored is not a perfect proxy for skill, but have drawn enough of a conclusion to warrant looking deeper. The data here suggests that betting odds account for home field advantage by depressing home betting odds.

Within our analysis, we recognize that there is no absence of confounding variables. We dig deeper to see if we can draw similar conclusions on a league by league basis.

In [20]:
```
MatchData["Home_Away_Diff"] = MatchData["home_team_goal"] - MatchData["away_te
am_goal"]
#We return to the original dataframe to prevent confusion and potential error
s.
#We create the same variable as our net_goal from above
```

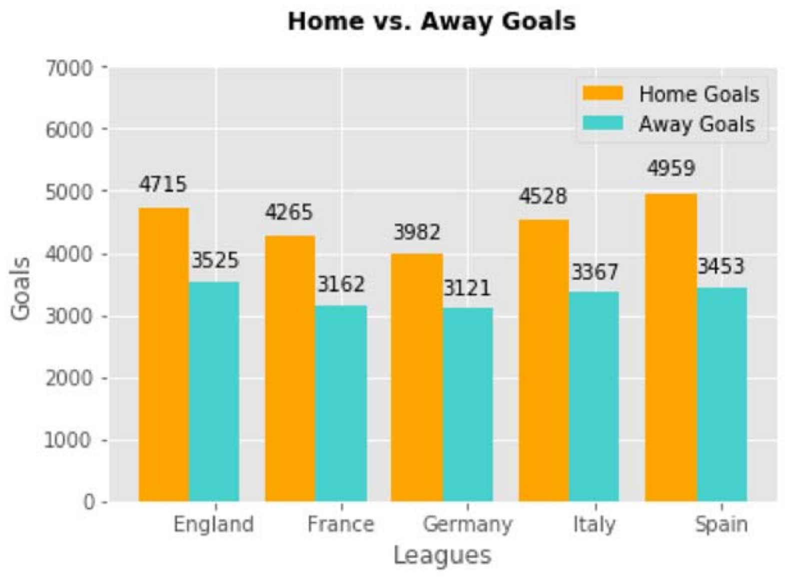
```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

In [21]:
```
def add_Result(row):
    if row["Home_Away_Diff"] > 0:
        return "Won"
    elif row["Home_Away_Diff"] == 0:
        return "Draw"
    elif row["Home_Away_Diff"] < 1:
        return "Lost"
    else:
        return "error"

MatchData = MatchData.assign(Result=MatchData.apply(add_Result, axis=1))
#We are again taking count of the number of games won, drawn, or lost for the
 home teams.
```

In [22]:
```
MatchData.set_index('league_id', inplace=True)
```

In [23]:
```
England = MatchData.loc[1729]
France = MatchData.loc[4769]
Germany = MatchData.loc[7809]
Italy = MatchData.loc[10257]
Spain = MatchData.loc[21518]

#We want to hone in on the leagues that have large fanbases and are likely to
 have high amounts of betting traffic.
```

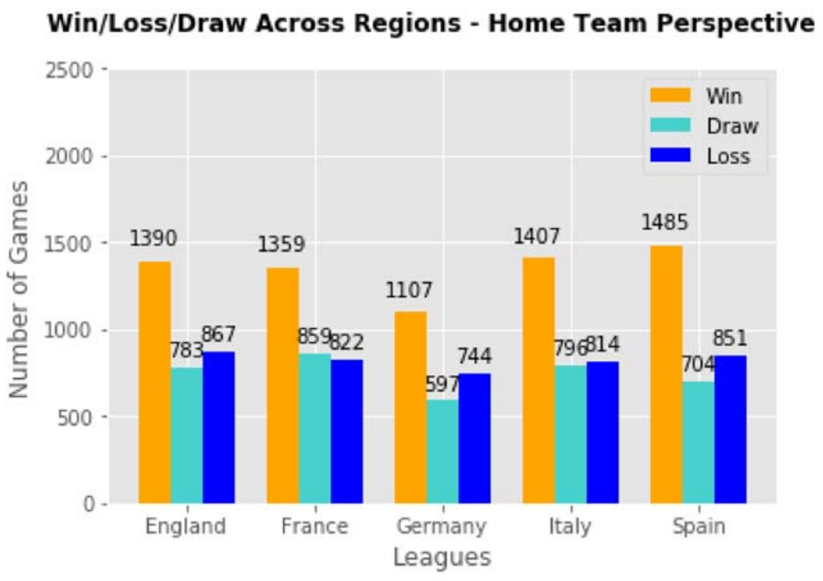
In [24]:
```
N = 5
ind = np.arange(N)
width = 0.4

fig, ax = plt.subplots()

home_vals = [England.home_team_goal.sum(), France.home_team_goal.sum(), G
ermany.home_team_goal.sum(), Italy.home_team_goal.sum(), Spain.home_team_
goal.sum()]
rects1 = ax.bar(ind, home_vals, width, color='orange')

away_vals = [England.away_team_goal.sum(), France.away_team_goal.sum(), G
ermany.away_team_goal.sum(), Italy.away_team_goal.sum(), Spain.away_team_
goal.sum()]
rects2 = ax.bar(ind+width, away_vals, width, color='mediumturquoise')

fig.suptitle("Home vs. Away Goals", fontweight = 'bold')

ax.set_ylim(0, 7000)
ax.set_ylabel('Goals')
ax.set_xlabel('Leagues')

ax.set_xticks(ind+width)
ax.set_xticklabels(('England', 'France', 'Germany', 'Italy', 'Spain'))
ax.legend((rects1[0], rects2[0]), ('Home Goals', 'Away Goals'))

def autolabel (rects):
    for rect in rects:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)
```



In this graph, we plotted the total home and away goals for each respective league that we chose to focus on to analyze whether or not there really is a home advantage and how each league did compared to the others. As shown above, its clear that there are significantly higher home goals across the leagues. Teams in the Spain LIGA BBVA league seem to perform the best on their own territory, followed by those in the England Premier League and Italy Serie A League. Interestingly, while there is a significant range of home goals across the leagues, the away goals are around the same number.

In [25]:
```
win_e, loss_e, draw_e, win_f, loss_f, draw_f, win_g, loss_g, draw_g, win_i, lo
ss_i, draw_i, win_s, loss_s, draw_s = (0,)*15

for row in England.Result:
    if row == "Draw":
        draw_e = draw_e + 1
    elif row =="Won":
        win_e = win_e + 1
    elif row == "Lost":
        loss_e = loss_e + 1

for row in France.Result:
    if row == "Draw":
        draw_f = draw_f + 1
    elif row =="Won":
        win_f = win_f + 1
    elif row == "Lost":
        loss_f = loss_f + 1

for row in Germany.Result:
    if row == "Draw":
        draw_g = draw_g + 1
    elif row =="Won":
        win_g = win_g + 1
    elif row == "Lost":
        loss_g = loss_g + 1

for row in Italy.Result:
    if row == "Draw":
        draw_i = draw_i + 1
    elif row =="Won":
        win_i = win_i + 1
    elif row == "Lost":
        loss_i = loss_i + 1

for row in Spain.Result:
    if row == "Draw":
        draw_s = draw_s + 1
    elif row =="Won":
        win_s = win_s + 1
    elif row == "Lost":
        loss_s = loss_s + 1
```

In [26]:
```
N = 5
ind = np.arange(N)
width = 0.25

fig, ax = plt.subplots()

yvals = [win_e, win_f, win_g, win_i, win_s]
r1 = ax.bar(ind, yvals, width, color='orange')

zvals = [draw_e, draw_f, draw_g, draw_i, draw_s]
r2 = ax.bar(ind+width, zvals, width, color='mediumturquoise')

kvals = [loss_e, loss_f, loss_g, loss_i, loss_s]
r3 = ax.bar(ind+width*2, kvals, width, color='b')

fig.suptitle("Win/Loss/Draw Across Regions - Home Team Perspective", fontweigh
t = 'bold')

ax.set_ylim(0, 2500)
ax.set_ylabel('Number of Games')
ax.set_xlabel('Leagues')

ax.set_xticks(ind+width)
ax.set_xticklabels(('England', 'France', 'Germany', 'Italy', 'Spain'))
ax.legend( (r1[0], r2[0], r3[0]), ('Win', 'Draw', 'Loss') )

def autolabel(r):
    for r in r:
        h = r.get_height()
        ax.text(r.get_x()+r.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')

autolabel(r1)
autolabel(r2)
autolabel(r3)

plt.show()
```



We also looked at the aggregate wins, losses, and draws in respects to the home team within each region to gain more insight into our analysis of the correlation between leagues and betting. In general, the home team wins more; the exceptions are Germany, which has a much smaller spread between wins and losses, and Spain, which has a much higher number of wins than losses.

England - 1 : 0.563 : 0.624

France - 1 : 0.632 : 0.605

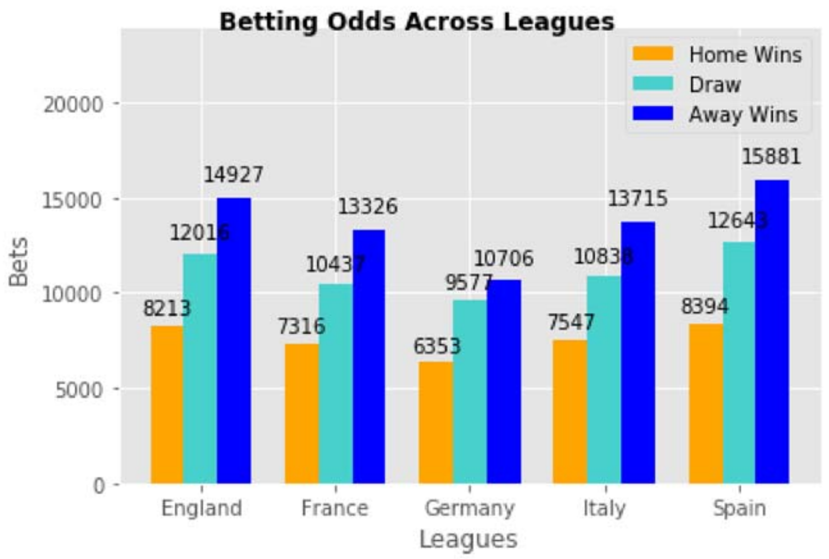
Germany - 1 : 0.539 : 0.672

Italy - 1 : 0.566 : 0.579

Spain - 1 : 0.474 : 0.573

The normalized ratios of wins : draws : loss for each league

In [27]:
```
N = 5
ind = np.arange(N)
width = 0.25

fig, ax = plt.subplots()

yvals = [England.B365H.sum(), France.B365H.sum(), Germany.B365H.sum(), It
aly.B365H.sum(), Spain.B365H.sum()]
r1 = ax.bar(ind, yvals, width, color='orange')
zvals = [England.B365D.sum(), France.B365D.sum(), Germany.B365D.sum(), It
aly.B365D.sum(), Spain.B365D.sum()]
r2 = ax.bar(ind+width, zvals, width, color='mediumturquoise')
kvals = [England.B365A.sum(), France.B365A.sum(), Germany.B365A.sum(), It
aly.B365A.sum(), Spain.B365A.sum()]
r3 = ax.bar(ind+width*2, kvals, width, color='b')

fig.suptitle("Betting Odds Across Leagues", fontweight = 'bold')

ax.set_ylim(0, 24000)
ax.set_ylabel('Bets')
ax.set_xlabel('Leagues')

ax.set_xticks(ind+width)
ax.set_xticklabels(('England', 'France', 'Germany', 'Italy', 'Spain'))
ax.legend( (r1[0], r2[0], r3[0]), ('Home Wins', 'Draw', 'Away Wins') )

def autolabel(r):
    for r in r:
        h = r.get_height()
        ax.text(r.get_x()+r.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')

autolabel(r1)
autolabel(r2)
autolabel(r3)
fig.tight_layout(w_pad=20)

plt.show()
```



England - 1 : 1.46 : 1.82

France - 1 : 1.43 : 1.82

Germany - 1 : 1.51 : 1.69

Italy - 1 : 1.44 : 1.82

Spain - 1 : 1.51 : 1.89

The normalized ratios for home : draw : away betting odds for each league.

Given the previous graphs and data, this one makes sense because given the notion of home advantage, there is a larger payout for the away team if they win. Out of all the leagues, Spain performed the best. This matches our betting odds outcome, with Spain having the highest spread between home and away win betting odds (highest payout if the away team wins because it has the lowest possbility of happening). We also see that the spread is smallest for Germany, which is justified as Germany has the smallest spread between wins and losses.

Betting Odds Conclusion

Based off of the league data, the negative correlation between home team advantage and betting odds is upheld. In terms of wins:losses, Spain had the largest spread, followed by Italy, France, England, and lastly, Germany. This pattern is reflected in terms of betting odds: Spain had the largest spread, followed by Italy/France/England, and lastly, Germany.

Investigating Individual Player Effects on Betting Odds

In [28]:
```
NameMatch= pd.Series(data = PlayerNames["player_name"].values, index = PlayerN
ames["player_api_id"].values)
TopPlayers["player_name"] = TopPlayers["player_api_id"].map(NameMatch)

# Matches player names from the Player Names Data Frame over to our merged dat
a frame using the map function
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
```

In [29]:
```
def best_by_player(df):
    return df.nlargest(1,"overall_rating") #shows highest possible ranking at
 a given moment in-time
```

In [30]:
```
match = TopPlayers.groupby("player_name")
new_df = pd.DataFrame(match.apply(best_by_player)) #creates a new data frame w
hich gives the highest score for each player
new_df.set_index("player_api_id") #set_index to player API Id
new_df.head(10)
```

Out[30]:

		player_fifa_api_id	player_api_id	overall_rating	preferred_foot	attac
player_name						
Aaron Lennon	**166**	152747	30895	84.0	right	high
Aaron Ramsey	**277**	186561	75489	83.0	right	high
Abdulkader Keita	**899**	157191	31012	82.0	right	medi
Adam Johnson	**1743**	165740	24159	82.0	left	high
Adam Lallana	**1792**	180819	37234	80.0	right	high
Adel Taarabt	**2352**	179605	47394	80.0	right	high
Adem Ljajic	**2400**	190544	155738	81.0	right	high
Aderlan Santos	**2418**	213374	361710	80.0	right	medi
Adil Rami	**2523**	183280	77741	84.0	right	high
Adrian Lopez	**2893**	173818	45744	81.0	right	high

10 rows × 40 columns

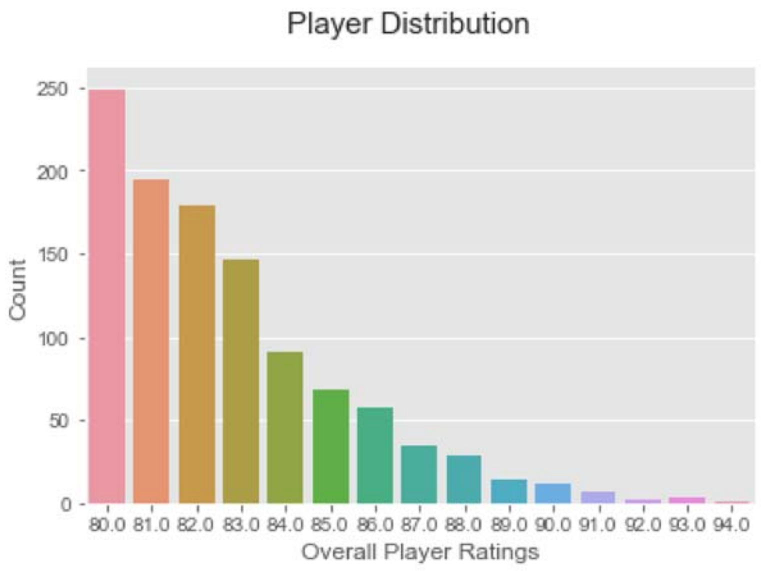
In [31]:

```
RatingScores = sns.countplot(x = new_df['overall_rating']) #creates the graph
 using the searborn package
sns.set_style('ticks')
sns.despine() #sets the spines off for the graph
plt.xlabel("Overall Player Ratings")
plt.ylabel("Count")
plt.suptitle("Player Distribution", fontsize = 15)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:1460: Futur
eWarning: remove_na is deprecated and is a private function. Do not use.
  stat_data = remove_na(group_data)
```

Out[31]:

```
Text(0.5,0.98,'Player Distribution')
```



In this portion of the project, we wanted to compare how top players are compared to other players. Lionel Messi has been a top player since 2008/2009, we benchmarked his average stats compared to other players, also called spread, to see how it best stacks up. Here, we tried establishing a correlation between Overall Rating and the average of the player attributes

In [32]:
```
cols = ['overall_rating','crossing', 'finishing', 'heading_accuracy', 'short_p
assing', 'volleys', 'dribbling', 'curve', 'free_kick_accuracy', 'long_passing'
, 'ball_control', 'acceleration', 'sprint_speed', 'agility', 'reactions', 'bal
ance', 'shot_power', 'jumping', 'stamina', 'strength', 'long_shots', 'aggressi
on', 'interceptions', 'positioning', 'vision', 'penalties', 'marking', 'standi
ng_tackle', 'sliding_tackle']
spread_df = new_df[cols] #finds information for select columns
spread_df["average"] = pd.DataFrame(spread_df.mean(axis=1))

x = float(spread_df["average"].loc[["Lionel Messi"]]) #in order to compare, we
 need to create this variable as a float, else the types don't match up

spread_df["Spread"] = pd.to_numeric(spread_df["average"])-x #benchmarks each p
layer to Lionel Messi's average
spread_df.head()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  import sys
```

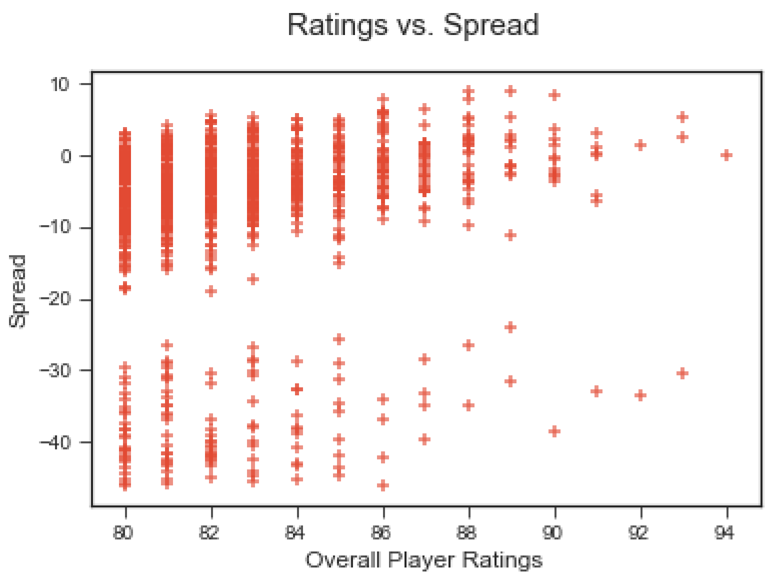
Out[32]:

		overall_rating	crossing	finishing	heading_accuracy	short_passing
player_name						
Aaron Lennon	**166**	84.0	82.0	70.0	38.0	77.0
Aaron Ramsey	**277**	83.0	74.0	77.0	58.0	84.0
Abdulkader Keita	**899**	82.0	75.0	72.0	77.0	73.0
Adam Johnson	**1743**	82.0	83.0	77.0	43.0	77.0
Adam Lallana	**1792**	80.0	74.0	73.0	67.0	83.0

5 rows × 31 columns

In [33]:
```
sns.regplot(x = spread_df['overall_rating'], y=spread_df['Spread'], fit_reg=Fa
lse, marker = "+")
plt.xlabel("Overall Player Ratings")
plt.ylabel("Spread")
plt.suptitle("Ratings vs. Spread", fontsize = 15)

#creates a scatterplot with markers that are an + using the seaborn package
```

Out[33]:
```
Text(0.5,0.98,'Ratings vs. Spread')
```



In [34]:
```
linregress(spread_df["overall_rating"],spread_df["Spread"]) #regresesion to fi
nd p and r value
```

Out[34]:
```
LinregressResult(slope=0.58689250160087603, intercept=-56.030921326252191, rv
alue=0.13204113142152654, pvalue=1.2685134607551335e-05, stderr=0.13381825383
633952)
```

Based on this, there is no statistical significance between player attributes and overall player ratings!

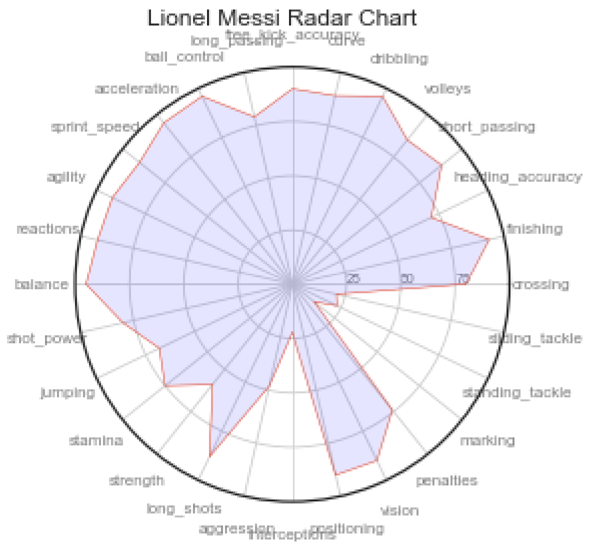
Another thing we wanted to do was to generate radar charts for top players to see how their attributes map out to other players. We created a radar chart for Lionel Messi and Cristiano Ronaldo.

In [35]:
```
labels = np.array(['crossing', 'finishing', 'heading_accuracy', 'short_passin
g', 'volleys', 'dribbling', 'curve', 'free_kick_accuracy', 'long_passing', 'ba
ll_control', 'acceleration', 'sprint_speed', 'agility', 'reactions', 'balance'
, 'shot_power', 'jumping', 'stamina', 'strength', 'long_shots', 'aggression',
'interceptions', 'positioning', 'vision', 'penalties', 'marking', 'standing_ta
ckle', 'sliding_tackle'])
stats = new_df.loc[["Lionel Messi"],labels].values[0]
#finds specific stats for Lionel Messi
```

```
In [36]: angles=np.linspace(0, 2*np.pi, len(labels), endpoint = False)  #dtermines numb
         er of angles it should break down into
         #close the plot
         stats = np.concatenate((stats,[stats[0]]))
         angles = np.concatenate((angles,[angles[0]]))

         #merges into one list of 28
```

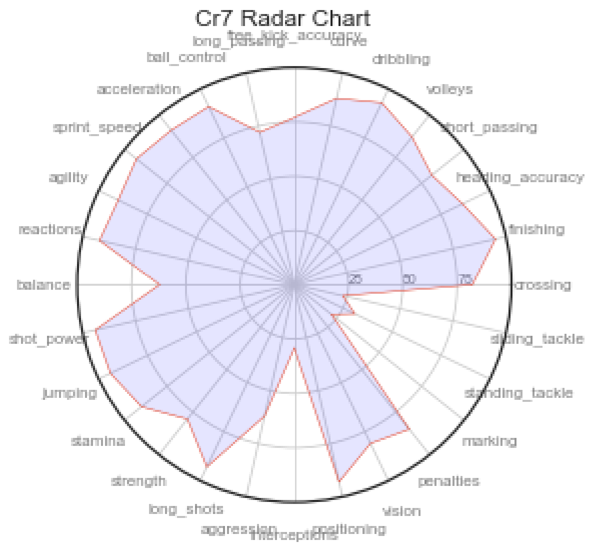
```
In [37]: fig = plt.figure()
         ax = fig.add_subplot(111, polar=True) #sets the graph as a polar graph
         plt.xticks(angles[:-1], labels, color='grey', size=8)
         ax.set_rlabel_position(0) #creates the graph (R-theta polar coordinate) from r
          = 0
         plt.yticks([25,50,75], ["25","50","75"], color="grey", size=7)
         plt.ylim(0,100)
         ax.plot(angles,stats, linewidth=.5, linestyle='solid')
         ax.fill(angles, stats, 'b', alpha=0.1) #fills in the graph
         ax.grid(True)


         plt.suptitle("Lionel Messi Radar Chart") #title
```

```
Out[37]: Text(0.5,0.98,'Lionel Messi Radar Chart')
```



In [38]:
```
labels = np.array(['crossing', 'finishing', 'heading_accuracy', 'short_passin
g', 'volleys', 'dribbling', 'curve', 'free_kick_accuracy', 'long_passing', 'ba
ll_control', 'acceleration', 'sprint_speed', 'agility', 'reactions', 'balance'
, 'shot_power', 'jumping', 'stamina', 'strength', 'long_shots', 'aggression',
'interceptions', 'positioning', 'vision', 'penalties', 'marking', 'standing_ta
ckle', 'sliding_tackle'])
stats = new_df.loc["Cristiano Ronaldo",labels].values[0]
angles=np.linspace(0, 2*np.pi, len(labels), endpoint = False)  #dtermines numb
er of angles it should break down into
#close the plot
stats = np.concatenate((stats,[stats[0]]))
angles = np.concatenate((angles,[angles[0]]))
fig = plt.figure()
ax = fig.add_subplot(111, polar=True)
plt.xticks(angles[:-1], labels, color='grey', size=8)
ax.set_rlabel_position(0)
plt.yticks([25,50,75], ["25","50","75"], color="grey", size=7)
plt.ylim(0,100)
ax.plot(angles,stats, linewidth=.5, linestyle='solid')
ax.fill(angles, stats, 'b', alpha=0.1)
ax.grid(True)


plt.suptitle("Cr7 Radar Chart")

#same thing what we did before for Lionel Messi for Cristiano Ronaldo
```

Out[38]:
```
Text(0.5,0.98,'Cr7 Radar Chart')
```



In [39]:
```
new_df.head()
```

Out[39]:

		player_fifa_api_id	player_api_id	overall_rating	preferred_foot	attac
player_name						
Aaron Lennon	**166**	152747	30895	84.0	right	high
Aaron Ramsey	**277**	186561	75489	83.0	right	high
Abdulkader Keita	**899**	157191	31012	82.0	right	medi
Adam Johnson	**1743**	165740	24159	82.0	left	high
Adam Lallana	**1792**	180819	37234	80.0	right	high

5 rows × 40 columns

In this section of the code, we found what team each player played for, allowing us to compile more team-based information. The purpose of this section was to see how the overall rating for a team corresponded with betting odds.

Problem 1: None of the excel files were linked together. For example, in this instance, we did not know what team each player played for. In this section, we map player to the team that they play for.

In [40]:
```
new_df["Team_Name"] = np.nan  #creates a new column with no entries - this all
ows us to fill it in later
variables = ["player_api_id","overall_rating","Team_Name"] #specifies what var
iables we are looking for
new_df = new_df[variables]
new_df.head(2)
```

Out[40]:

		player_api_id	overall_rating	Team_Name
player_name				
Aaron Lennon	**166**	30895	84.0	NaN
Aaron Ramsey	**277**	75489	83.0	NaN

In [41]:
```
#i = 0
TeamName = []


#for PlayerId in new_df.player_api_id:
for pInd, pRow in new_df.iterrows(): #goes through our data frame row by
 row
    PlayerId = pRow.player_api_id #sets player value to match for
    found = False
    for index,row in MatchData.iterrows():  #row by row in the MatchData
 file
        for y in range(1,12): #goes through the columns for Home_Player_1
 to Home_Player_12
            if row["home_player_"+str(y)] == PlayerId:
                new_df.at[pInd,"Team_Name"] = row["home_team_api_id"] #if
 there's a match, add that value to that specfic cell and break
                print("matched")
                found = True
                break
        if found == True:
            break
    #if i > 5:
     #   break
    #i += 1
new_df.head(2)


new_df.shape
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
```

Out[41]:

```
(1086, 3)
```

In [42]:

```
TeamMatch= pd.Series(data = TeamData["team_long_name"].values, index = TeamDat
a["team_api_id"].values)
new_df["Team_Long_Name"] = new_df["Team_Name"].map(TeamMatch)


#assigns team names
```

In [43]:

```
team_df = new_df.groupby("Team_Long_Name").mean() #consolidates all the teams
 and finds average of the overall_rating
team_df = team_df.reset_index()
team_df.head(10)
```

Out[43]:

	Team_Long_Name	player_api_id	overall_rating	Team_Name
0	1. FC Köln	94800.400000	82.600000	8722.0
1	1. FC Nürnberg	77897.000000	82.666667	8165.0
2	1. FSV Mainz 05	156752.500000	80.000000	9905.0
3	AC Ajaccio	35179.000000	83.500000	8576.0
4	AC Arles-Avignon	41464.000000	83.000000	108893.0
5	AJ Auxerre	33381.000000	82.333333	8583.0
6	AS Monaco	98574.933333	82.133333	9829.0
7	AS Saint-Étienne	29062.428571	81.571429	9853.0
8	Ajax	37643.000000	83.000000	8593.0
9	Arsenal	41428.314286	83.800000	9825.0

Problem 3: In this portion of the code, we create a new data frame for Bet 365 Betting odds. We create a new column, called spread, which calculates the difference between home price and away/draw odds. Since this is a new data frame from a seperate CSV file, we had to once again map it to the team names as well.

In [44]:

```
MatchDataF = MatchData1[["home_team_api_id","B365H","B365D","B365A"]] #new dat
a frame that extracts value from the MatchData CSV
MatchDataF["B.Spread"] = MatchDataF["B365H"] - MatchDataF["B365D"] - MatchData
F['B365A'] #the more negative, the stronger oddsmaker price them
MatchDataF.groupby("home_team_api_id").mean() #finds the mean for specific var
iables team by team

MatchDataF.shape

TeamMatch= pd.Series(data = TeamData["team_long_name"].values, index = TeamDat
a["team_api_id"].values)
MatchDataF["Team_Unique_Name"] = MatchDataF["home_team_api_id"].map(TeamMatch)
 #matches team names with team values

cols = ["Team_Unique_Name","home_team_api_id","B365H","B365D","B365A","B.Sprea
d"]


MatchDataF = MatchDataF[cols]
MatchDataF.sort_values(by = ["Team_Unique_Name"]) #sorts alphabetically
MatchDataF.head()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
```

Out[44]:

	Team_Unique_Name	home_team_api_id	B365H	B365D	B365A	B.Spread
0	KRC Genk	9987	1.73	3.40	5.00	-6.67
1	SV Zulte-Waregem	10000	1.95	3.20	3.60	-4.85
2	KSV Cercle Brugge	9984	2.38	3.30	2.75	-3.67
3	KAA Gent	9991	1.44	3.75	7.50	-9.81
4	FCV Dender EH	7947	5.00	3.50	1.65	-0.15

In [45]:
```
for pInd, pRow in team_df.iterrows(): #matches values one from data frame to t
he other
    TeamId = pRow.Team_Name
    found = False
    for index,row in MatchDataF.iterrows():  #rowbyrow
        if row["home_team_api_id"] == TeamId: # if there is a match, then add
 a new column to our original dataframe
            team_df.at[pInd,"B365H"] = row["B365H"]
            team_df.at[pInd,"B365D"] = row["B365D"]
            team_df.at[pInd,"B365A"] = row["B365A"]
            team_df.at[pInd,"B.Spread"] = row["B.Spread"]
            print("matched")
            found = True
            break

    #if i > 5:
     #   break
    #i += 1
team_df.head(155)
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

```
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
matched
```

Out[45]:

	Team_Long_Name	player_api_id	overall_rating	Team_Name	B365H	B365D	B365
0	1. FC Köln	94800.400000	82.600000	8722.0	2.62	3.30	2.60
1	1. FC Nürnberg	77897.000000	82.666667	8165.0	3.50	3.30	2.10
2	1. FSV Mainz 05	156752.500000	80.000000	9905.0	3.80	3.40	2.00
3	AC Ajaccio	35179.000000	83.500000	8576.0	2.63	3.00	2.88
4	AC Arles-Avignon	41464.000000	83.000000	108893.0	7.00	4.00	1.50
5	AJ Auxerre	33381.000000	82.333333	8583.0	2.10	3.10	3.75
6	AS Monaco	98574.933333	82.133333	9829.0	2.40	3.10	3.10
7	AS Saint-Étienne	29062.428571	81.571429	9853.0	1.73	3.50	5.00
8	Ajax	37643.000000	83.000000	8593.0	1.17	7.50	13.0
9	Arsenal	41428.314286	83.800000	9825.0	1.20	6.50	15.0
10	Aston Villa	30764.307692	82.538462	10252.0	1.91	3.40	4.33
11	Atalanta	53549.200000	81.400000	8524.0	2.05	3.10	4.00
12	Athletic Club de Bilbao	133278.400000	80.900000	8315.0	2.00	3.30	3.80
13	Atlético Madrid	94567.666667	82.333333	9906.0	1.44	4.20	7.50
14	Bari	53471.600000	81.800000	9976.0	1.85	3.30	4.50
15	Bayer 04 Leverkusen	75342.266667	82.533333	8178.0	2.05	3.30	3.60
16	Birmingham City	32541.800000	81.800000	8658.0	2.90	3.25	2.50
17	Blackburn Rovers	50281.750000	83.125000	8655.0	8.00	4.33	1.44
18	Blackpool	39109.000000	80.000000	8483.0	2.90	3.25	2.50
19	Bologna	38460.666667	81.666667	9857.0	5.25	3.25	1.75
20	Bolton Wanderers	58326.666667	81.777778	8559.0	1.83	3.50	4.50
21	Borussia Dortmund	61362.000000	82.466667	9789.0	1.55	3.80	6.25
22	Borussia Mönchengladbach	94320.818182	82.090909	9788.0	2.75	3.30	2.50
23	Brescia	39762.000000	80.000000	9858.0	4.75	3.30	1.80
24	CA Osasuna	33504.000000	81.333333	8371.0	2.80	3.30	2.50
25	CF Os Belenenses	78544.000000	80.000000	9807.0	2.50	3.00	2.80
26	Cagliari	53407.000000	81.600000	8529.0	3.00	3.00	2.50
27	Cardiff City	49970.000000	81.000000	8344.0	3.40	3.40	2.30
28	Catania	28728.000000	80.000000	8530.0	2.40	3.00	3.20
29	Celtic	31946.142857	84.000000	9925.0	1.17	6.50	17.0

	Team_Long_Name	player_api_id	overall_rating	Team_Name	B365H	B365D	B365
...	...	...	...	...	...	...	...
125	Sint-Truidense VV	37868.000000	82.000000	9997.0	2.50	3.40	2.70
126	Southampton	79177.416667	80.916667	8466.0	2.38	3.50	2.88
127	Sporting CP	110553.166667	81.166667	9768.0	1.25	5.50	15.0
128	Stade Rennais FC	47411.285714	80.857143	9851.0	2.70	3.00	2.80
129	Stade de Reims	92566.000000	81.000000	9837.0	3.50	3.10	2.20
130	Standard de Liège	82875.200000	81.800000	9985.0	1.30	5.25	9.50
131	Stoke City	72814.600000	80.600000	10194.0	2.60	3.30	2.75
132	Sunderland	48187.307692	81.692308	8472.0	5.50	3.60	1.67
133	Swansea City	59052.625000	81.000000	10003.0	2.10	3.30	3.60
134	TSG 1899 Hoffenheim	101279.800000	81.200000	8226.0	1.50	4.00	6.50
135	Torino	98695.000000	81.571429	9804.0	1.80	3.10	5.25
136	Tottenham Hotspur	59712.200000	82.880000	8586.0	3.50	3.30	2.10
137	Toulouse FC	70859.750000	82.250000	9941.0	2.10	3.10	3.75
138	UD Almería	62108.666667	81.000000	9865.0	2.00	3.30	3.80
139	Udinese	67990.333333	81.750000	8600.0	2.10	3.10	3.80
140	União de Leiria, SAD	177126.000000	82.000000	9771.0	2.20	3.20	3.40
141	Valencia CF	74922.923077	82.000000	10267.0	1.70	3.60	5.25
142	Valenciennes FC	26157.000000	81.000000	9873.0	2.40	3.10	3.10
143	VfB Stuttgart	38633.818182	83.454545	10269.0	1.53	3.80	6.50
144	VfL Bochum	28885.000000	81.000000	9911.0	3.20	3.40	2.20
145	VfL Wolfsburg	44100.400000	82.300000	8721.0	1.62	3.60	6.00
146	Villarreal CF	78879.363636	82.090909	10205.0	1.53	3.75	7.00
147	Vitesse	30335.000000	80.000000	8277.0	2.20	3.30	3.00
148	Vitória Guimarães	179130.000000	80.000000	7844.0	2.10	3.25	3.60
149	Vitória Setúbal	300916.000000	80.000000	10238.0	4.33	3.50	1.73
150	Watford	37521.000000	80.000000	9817.0	2.60	3.40	2.90
151	West Bromwich Albion	37087.571429	81.714286	8659.0	2.50	3.30	2.88
152	West Ham United	40018.722222	81.500000	8654.0	1.91	3.40	4.20
153	Wigan Athletic	55221.833333	80.666667	8528.0	1.80	3.40	5.00

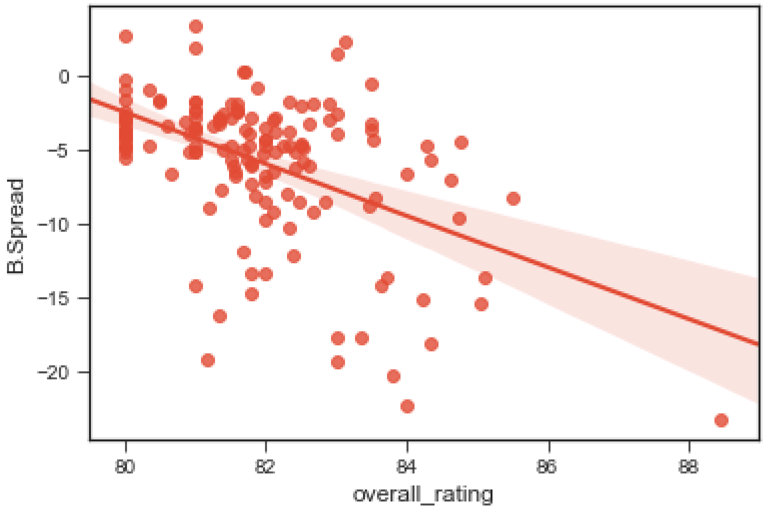
	Team_Long_Name	player_api_id	overall_rating	Team_Name	B365H	B365D	B36
154	Xerez Club Deportivo	49836.000000	80.000000	9868.0	2.80	3.25	2.50

155 rows × 8 columns

Here, we create a scatter plot to determine if there was any sort of correlation or not.

```
In [46]: sns.regplot(x = team_df['overall_rating'], y=team_df['B.Spread']) #regression
          using Seaborn package
```

```
Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x1fb5648a208>
```



Based on this scatterplot, we see a very slight correlation that exists between a team's overall rating and the betting spreads. The reason that we have hypothesized that such is the case is due to the disparity in the leagues. For example, certain leagues, despite being weaker than their international counterparts, have relatively more consolidated leagues where the winner is predictable. As a result, betting in those leagues means that their spread will always be much more negative, despite being fundamentally a worse team based on their team's overall rating compared to teams in other leagues. In short, the analysis is likely clouded by ommitted variables and a concrete conclusion is difficult with the data used.

Source

https://www.kaggle.com/hugomathien/soccer/data (https://www.kaggle.com/hugomathien/soccer/data)