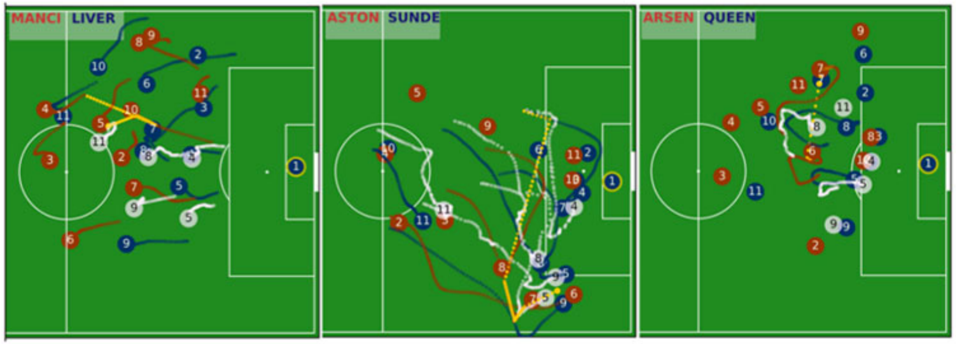
Soccer Predictions

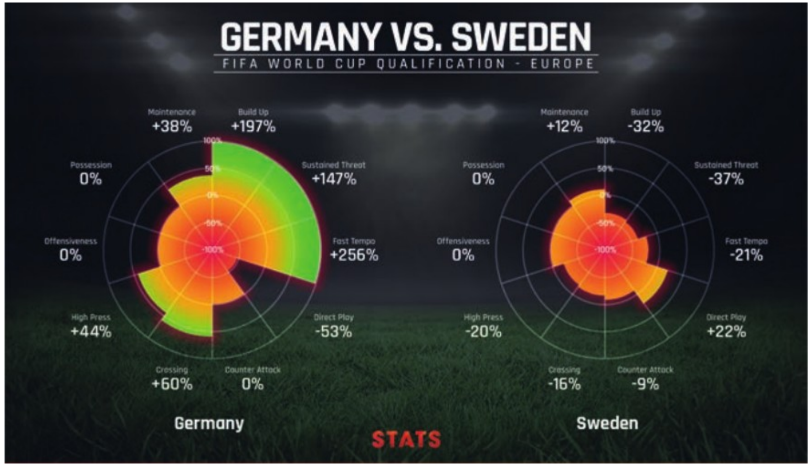
Soccer Predictions

By Domingo Imperatori

Project Goal

- Understand how predictable is Soccer, and what are some of the features that have influence in the matches outcome.





Data

Data from Kaggle

* kaggle:

https://www.kaggle.com/mkhvalchik/soccer/data

https://www.kaggle.com/hugomathien/soccer

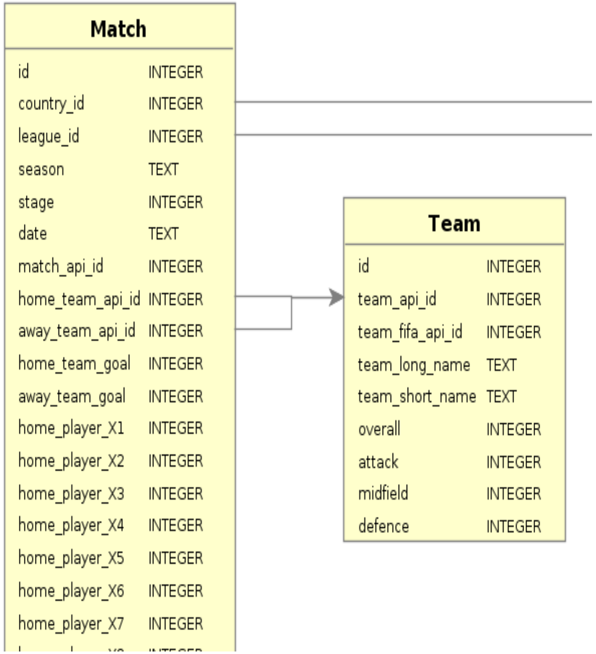
data of:

+25,000 matches

+10,000 players
11 European Countries with their lead championship
Seasons 2008 to 2016
Players and Teams' attributes* sourced from EA Sports' FIFA video game series, including the weekly updates
Team line up with squad formation (X, Y coordinates)
Betting odds from up to 10 providers
Detailed match events (goal types, possession, corner, cross, fouls, cards etc...) for +10,000 matches.
* Other Databases.

Data

- We will focus in 2 tables:
  - Matches: more than 25,000 matches European leagues
  - Team (299 teams)



Team

We scrap some data from www.sofifa.com

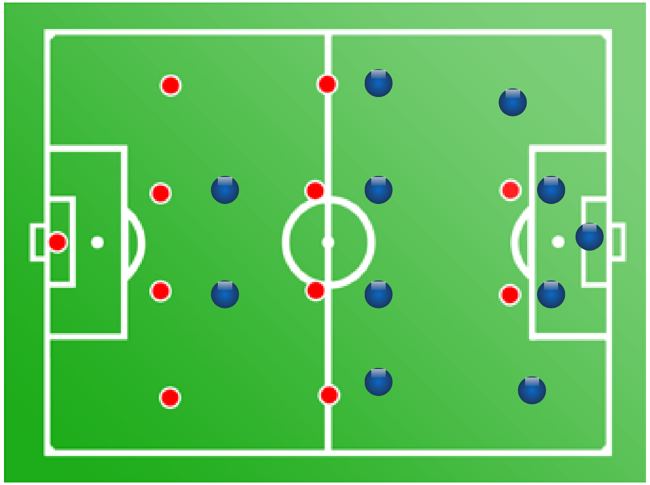
Specifically:

- Overall: how good is the team in total
- Attack: how good is attacking
- Midfield: how good is team midfield
- Defence: how good is team defense

Feature engineering

After importing and cleaning data.

- Differences between the different team crossing lines
  - example: home team attacking,visitor team defending



att_vs_def mid_vs_mid def_vs_att

In [31]:

```
# Overall differences between teams
matches['overall']=matches['home_overall']-matches['away_overall']

#Differences between the differents lines playing
matches['hatt_vs_adef']=matches['home_attack']-matches['away_defence']
matches['hdef_vs_aatt']=matches['home_defence']-matches['away_attack']
matches['hmid_vs_amid']=matches['home_midfield']-matches['away_midfield']
```

Column win

Create column win from matches (score: home_team_goal,away_team_goal):

- 0 : Home Team wins
- 1 : Visitor Team wins
- 2 : Draw

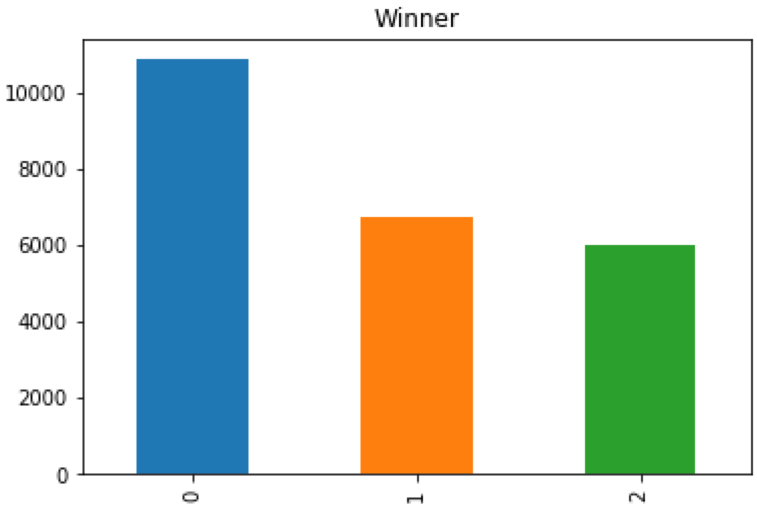
Create a new win column: 0-home team won 1-visitor team won 2-Tie

```
In [734]: matches['win']=np.where((matches['home_team_goal'] - matches['away_team_goal'])>0,0,-1)
          matches['win']=np.where((matches['home_team_goal'] - matches['away_team_goal'])<0,1,matches['win'])
          matches['win']=np.where((matches['home_team_goal'] - matches['away_team_goal'])==0,2,matches['win'])
          matches['win'].head()

Out[734]: 0    2
          1    2
          2    0
          3    2
          4    0
          Name: win, dtype: int64
```

Home Factor

- The home team winning is clearly the option most probable, second visitor winning (around 17% less) and finally draw the less probable



In [737]:
```
print("Percentage Home Wins: {0:.2%}".format(matches[matches['win']==0].win.count()/matches['win'].value_counts().su
print("Percentage Visitor Wins: {0:.2%}".format(matches[matches['win']==1].win.count()/matches['win'].value_counts()
print("Percentage Tie: {0:.2%}".format(matches[matches['win']==2].win.count()/matches['win'].value_counts().sum()))
```

```
Percentage Home Wins: 46.02%
Percentage Visitor Wins: 28.64%
Percentage Tie: 25.34%
```

Goal

Logically the goal is a decisive part of the game, so the team that score the most and receive the less got clearly more chances of winning.

Engineering features from goal:

- Mean of goals score by team in all the matches.
- Mean of goals received by team in all the matches.
- Effectiveness: difference between the mean of goals score and the goals received.

Created this features for playing as home and playing as visitor.

Describe matches:

```
In [52]: home_attack','home_midfield','home_defence','away_overall','away_attack','away_midfield','away_defence']].describe()
```

Out[52]:

	home_overall	home_attack	home_midfield	home_defence	away_overall	away_attack	away_midfield	away_defence
count	23575.000000	23575.000000	23575.000000	23575.000000	23575.000000	23575.000000	23575.000000	23575.000000
mean	72.735949	73.275080	72.670626	72.070541	72.735270	73.274231	72.669183	72.069777
std	5.839941	6.387009	6.179148	6.104974	5.841167	6.388142	6.181631	6.106442
min	58.000000	55.000000	56.000000	55.000000	58.000000	55.000000	56.000000	55.000000
25%	68.000000	69.000000	68.000000	67.000000	68.000000	69.000000	68.000000	67.000000
50%	73.000000	73.000000	73.000000	72.000000	73.000000	73.000000	73.000000	72.000000
75%	77.000000	78.000000	77.000000	76.000000	77.000000	78.000000	77.000000	76.000000
max	86.000000	91.000000	87.000000	86.000000	86.000000	91.000000	87.000000	86.000000

Describe 2:

```
matches[['home_team_mean','away_team_mean','home_rec_mean','away_rec_mean','home_effec','away_effec','win']].c
```

	home_team_mean	away_team_mean	home_rec_mean	away_rec_mean	home_effec	away_effec	win
count	23575.000000	23575.000000	23575.000000	23575.000000	23575.000000	23575.000000	23575.000000
mean	1.545748	1.159067	1.159067	1.545825	0.386681	-0.386758	0.793128
std	0.441231	0.328585	0.263302	0.441142	0.624709	0.217490	0.819041
min	0.533333	0.263158	0.550459	0.533333	-1.529412	-1.105263	0.000000
25%	1.261905	0.940789	0.982301	1.261905	-0.037383	-0.526316	0.000000
50%	1.445652	1.087719	1.157895	1.451128	0.289474	-0.377483	1.000000
75%	1.760331	1.308824	1.326087	1.760331	0.676471	-0.230263	2.000000
max	3.322368	2.328947	2.210526	3.322368	2.592105	0.470588	2.000000

Barcelona best Team:

```python
# Now matematically I can demonstrate that Barcelona is the best team at least at home:
matches[['home_team_name','home_effec']].groupby('home_team_name').mean().sort_values(by='home_effec',ascending=Fals
```

	home_effec
home_team_name	
FC Barcelona	2.592105
Real Madrid CF	2.355263
FC Bayern Munich	2.102941
SL Benfica	1.990826
Celtic	1.985507

Exploratory Data Analysis

- The problem is a classification type: we will try to figured out the match outcome using the different features :
  - Home Win: 0
  - Home Lost: 1
  - Draw: 2

EDA 1

- The first thing we can do is ordered by some of the variables for example

```
In [66]: #Sort by home_effec
         matches[['home_effec','win']].sort_values(by='home_effec')
```

Out[66]:

	home_effec	win
8670	-1.529412	1
8720	-1.529412	1
8874	-1.529412	1
8737	-1.529412	1
8806	-1.529412	2
8704	-1.529412	2
8771	-1.529412	1
8687	-1.529412	1
8754	-1.529412	1
8619	-1.529412	2
8635	-1.529412	1
8823	-1.529412	1
8857	-1.529412	1
8840	-1.529412	1
8891	-1.529412	1
8907	-1.529412	1
8788	-1.529412	2
11455	-1.421053	1
11625	-1.421053	1
11644	-1.421053	0
11606	-1.421053	1
11341	-1.421053	1
11683	-1.421053	1
11664	-1.421053	1

...	...	...
19483	2.592105	0
19905	2.592105	0
20893	2.592105	0
21257	2.592105	0
21453	2.592105	2
19484	2.592105	0
20646	2.592105	0
19673	2.592105	2
21889	2.592105	0
20185	2.592105	0
20187	2.592105	0
22391	2.592105	0
22022	2.592105	0
19835	2.592105	0
19836	2.592105	0
21124	2.592105	0
21566	2.592105	0
21605	2.592105	0
19862	2.592105	0
20798	2.592105	0
19674	2.592105	0
21736	2.592105	0
22372	2.592105	0
22334	2.592105	0
21870	2.592105	0
21434	2.592105	0
21275	2.592105	0
20874	2.592105	0
20186	2.592105	0
19958	2.592105	0

23575 rows × 2 columns

We can see that the effectiveness at home can be a good variable to discriminate:

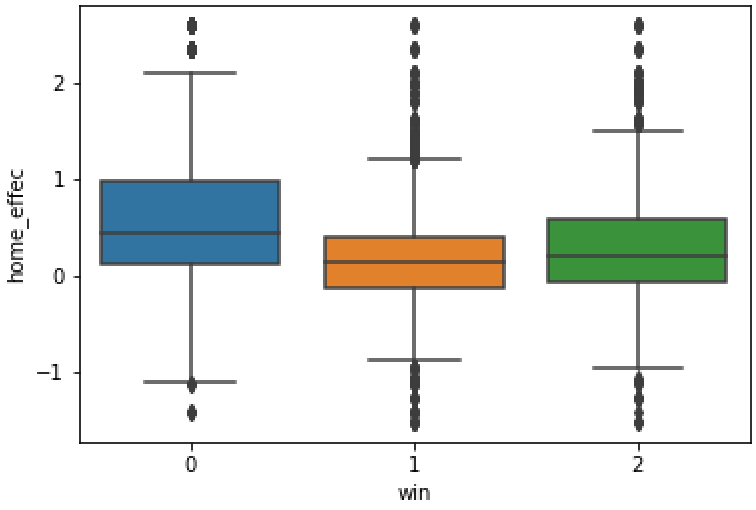
Low Effec: means more 1 (Visitor wins)

High Effec: means more 0 (Home wins)

EDA 2

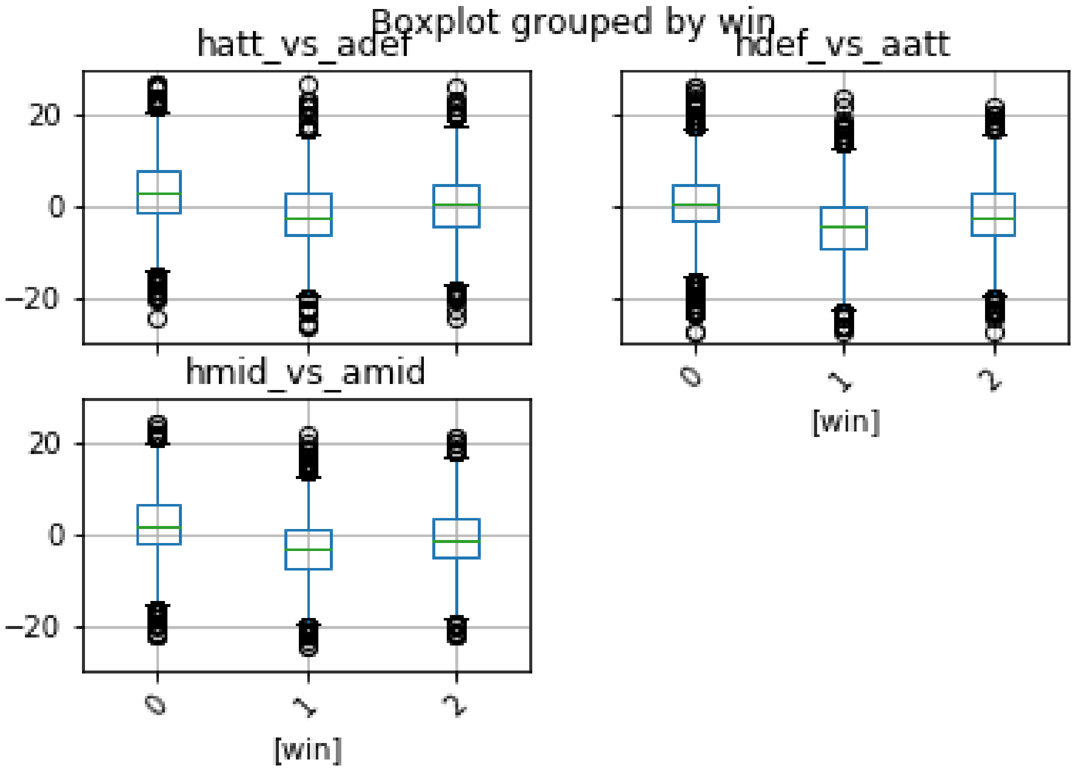
In [69]:
```
#We can see how is mean for the different outcomes (Win)
matches[['home_effec','win']].groupby('win',axis=0).home_effec.mean()
```

Out[69]:
```
win
0    0.577047
1    0.158138
2    0.299227
Name: home_effec, dtype: float64
```

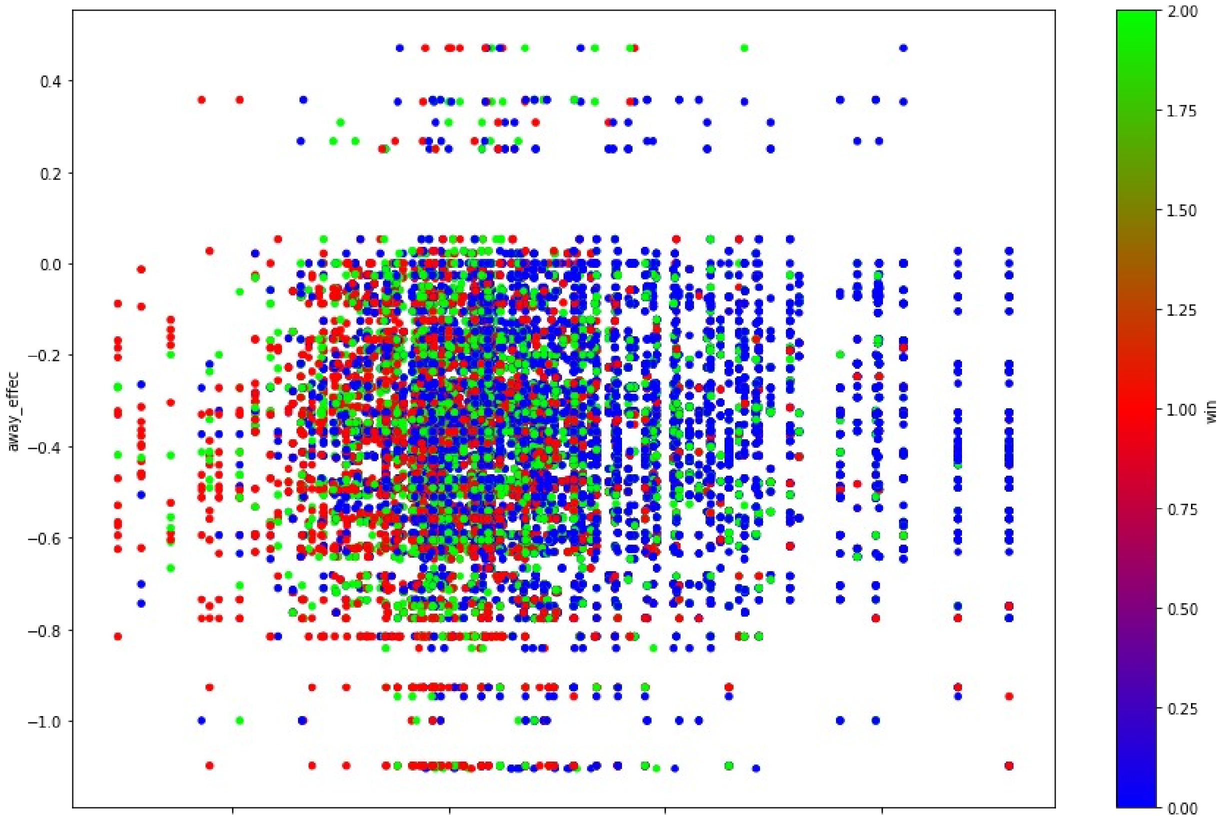


In [73]:
```
# let's look at this  graphically using boxplot:
sns.boxplot(x="win",y="home_effec",data=matches);
```

EDA 3


Boxplot grouped by win
hatt_vs_adef
hdef_vs_aatt
hmid_vs_amid
20
0
−20
20
0
−20
0
1
2
[win]
0
1
2
[win]

EDA 4

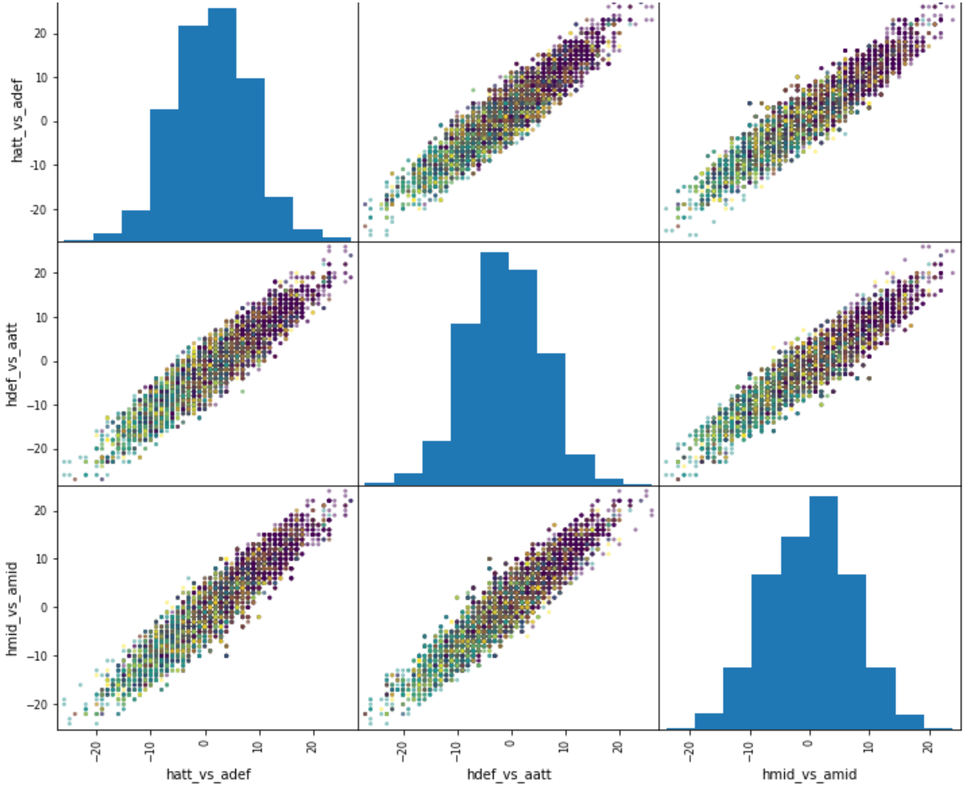


In [85]:

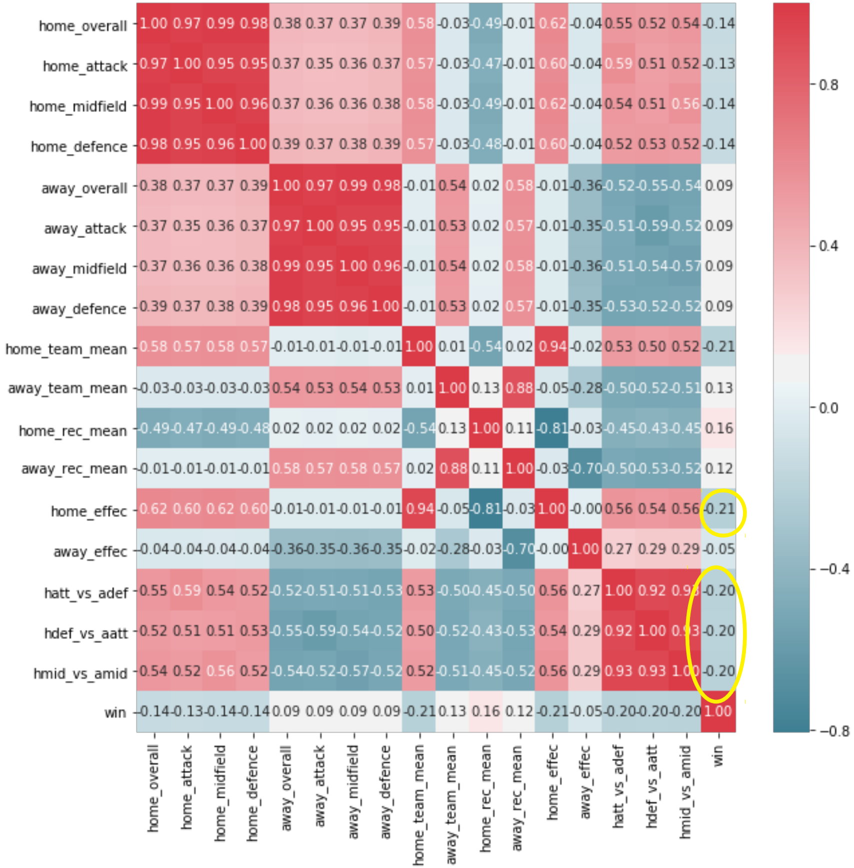
```
matches.plot(kind='scatter',
             x='home_effec',
             y='away_effec',
             c='win',
             colormap='brg',figsize=(15,10));
```

EDA 5

```
In [90]: pd.plotting.scatter_matrix(matches[['hatt_vs_adef','hdef_vs_aatt','hmid_vs_amid','win']].drop('win', axis=1),
                                     c=matches[['hatt_vs_adef','hdef_vs_aatt','hmid_vs_amid','win']].win, figsize=(12, 10));
```



EDA 6



In [91]:

```
matches_correlations = matches_ref.corr();
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(matches_correlations, cmap=cmap, annot=True, fmt='.2f')
plt.gcf().set_size_inches(10, 10);
```

Models

Let’s check some models with the variables we choose:

- K-Nearest Neighbors
- Logistic Regression
- Random Forest

K-Nearest Neighbors

K-Nearest Neighbours

In [163]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [166]:
```
X=matches[['home_effec','hatt_vs_adef','hdef_vs_aatt','hmid_vs_amid','home_team_mean']]
y=matches['win']
```

In [167]:
```
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=99)

#Feature Scaling (Standarize)
from sklearn.preprocessing import StandardScaler
sc_X=StandardScaler()
X_train=sc_X.fit_transform(X_train)
X_test=sc_X.transform(X_test)
```

KNN-2

In [170]:
```
for i in range(10,500,10):
    #k=50
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)


    y_pred_class = knn.predict(X_test)
    print("k=",i)
    print((metrics.accuracy_score(y_test, y_pred_class)))
```

```
k= 10
0.47964031218187986
k= 20
0.4884628435697319
k= 30
0.502035968781812
k= 40
0.5030539531727181
k= 50
0.504580929759077
k= 60
0.505938242280285
k= 70
0.5128944689514761
k= 80
0.5111978282999661
```

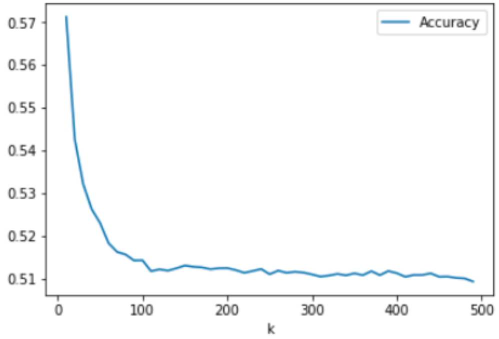
```
k= 90
0.5142517814726841
k= 100
0.513742789277231
k= 110
0.5147607736681371
k= 120
0.5167967424499491
k= 130
0.5181540549711571
k= 140
0.5181540549711571
k= 150
0.5184933831014591
k= 160
0.5186630471666102
k= 170
0.5203596878181201
k= 180
0.5203596878181201
k= 190
0.5183237190363081
k= 200
0.5196810315575161
k= 210
0.5195113674923652
k= 220
0.5174753987105531
k= 230
0.5166270783847982
k= 240
0.5174753987105531
k= 250
0.5188327112317611
k= 260
0.5198506956226672
k= 270
0.5217170003393281
k= 280
0.5206990159484222
k= 290
0.5213776722090261
k= 300
0.5210383440787242
k= 310
0.5210383440787242
```

KNN-3

```
In [172]: knn.predict_proba(X)
Out[172]: array([[0.76896552, 0.07586207, 0.15517241],
                 [0.76896552, 0.07586207, 0.15517241],
                 [0.76896552, 0.07586207, 0.15517241],
                 ...,
                 [0.34827586, 0.43448276, 0.21724138],
                 [0.17241379, 0.57931034, 0.24827586],
                 [0.17241379, 0.57931034, 0.24827586]])
```

```
In [173]: accuracies = []
          for k in range(10,500,10):
              knn = KNeighborsClassifier(n_neighbors=k)
              knn.fit(X,y)
              pred = knn.predict(X)
              accuracy = float(sum(pred == y)) / len(y)
              accuracies.append([k, accuracy])
```

```
In [174]: data = pd.DataFrame(accuracies,columns=['k','Accuracy'])
          data.plot.line(x='k',y='Accuracy');
```




```
In [182]: # Calculate TRAINING ERROR and TESTING ERROR for K=1 through 100.

          k_range = list(range(10,500,2))
          training_error = []
          testing_error = []

          # Find test accuracy for all values of K between 1 and 100 (inclusive).
          for k in k_range:

              # Instantiate the model with the current K value.
              knn = KNeighborsClassifier(n_neighbors=k)
              knn.fit(X_train, y_train)

              # Calculate training error (error = 1 - accuracy).
              y_pred_class = knn.predict(X_train)
              training_accuracy = metrics.accuracy_score(y_train, y_pred_class)
              training_error.append(1 - training_accuracy)

              # Calculate testing error.
              y_pred_class = knn.predict(X_test)
              testing_accuracy = metrics.accuracy_score(y_test, y_pred_class)
              testing_error.append(1 - testing_accuracy)
```

```
In [183]: # Allow plots to appear in the notebook.
          %matplotlib inline
          import matplotlib.pyplot as plt
          plt.style.use('fivethirtyeight')
```

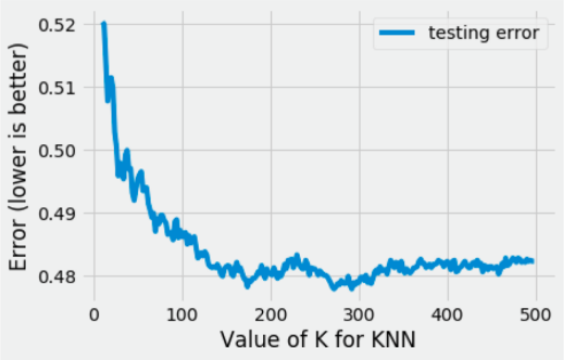
```
In [184]: # Create a DataFrame of K, training error, and testing error.
          column_dict = {'K': k_range, 'training error':training_error, 'testing error':testing_error}
          df = pd.DataFrame(column_dict).set_index('K').sort_index(ascending=False)
          df.head()
```

Out[184]:

K	testing error	training error
498	0.482355	0.489113
496	0.482355	0.489282
494	0.482525	0.489056
492	0.482355	0.489169
490	0.482694	0.489169

KNN-4

In [185]:
```
# Plot the relationship between K (HIGH TO LOW) and TESTING ERROR.
df.plot(y='testing error');
plt.xlabel('Value of K for KNN');
plt.ylabel('Error (lower is better)');
```



In [186]:
```
# Find the minimum testing error and the associated K value.
df.sort_values('testing error').head()
```

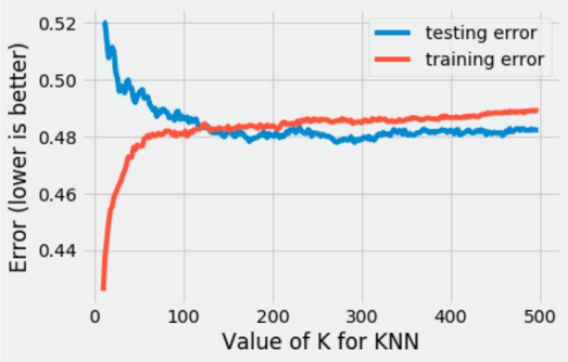
Out[186]:

K	testing error	training error
292	0.477944	0.485097
272	0.477944	0.485606
270	0.478283	0.485549
274	0.478283	0.485776
174	0.478283	0.484022

In [187]:
```
# Alternative method:
min(list(zip(testing_error, k_range)))
```

Out[187]: (0.4779436715303699, 272)

In [188]:
```
# Plot the relationship between K (HIGH TO LOW) and both TRAINING ERROR and TESTING ERROR.
df.plot();
plt.xlabel('Value of K for KNN');
plt.ylabel('Error (lower is better)');
```



In [193]:
```
knn=KNeighborsClassifier(n_neighbors=272)
knn.fit(X_train,y_train)
y_pred_class = knn.predict(X_test)
print((metrics.accuracy_score(y_test, y_pred_class)))
```

0.5220563284696301

The best Predictions is: 52% (k=272)

Logistic Regression

```
In [198]: #Fitting Logistic Reegresion to the Training set
          from sklearn.linear_model import LogisticRegression
          classifier= LogisticRegression()
          classifier.fit(X_train,y_train)

Out[198]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                    verbose=0, warm_start=False)

In [199]: #Predicting the Test set results
          y_pred= classifier.predict(X_test)

In [200]: print((metrics.accuracy_score(y_test, y_pred)))

          0.5159484221241941
```

The best Predictions is: 51% (almost 52)

Random Forest

Random Forest

In [51]:
```
###### Fitting Random Forest Classidication to the training set
from sklearn.ensemble import RandomForestClassifier

X=matches[['home_overall','home_attack','home_midfield','home_defence','away_overall','away_attack','away_midfield',
y=matches['win']

X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=99)

#Feature Scaling (Standarize)
from sklearn.preprocessing import StandardScaler
sc_X=StandardScaler()
X_train=sc_X.fit_transform(X_train)
X_test=sc_X.transform(X_test)

classifier= RandomForestClassifier(n_estimators=500, max_features=12,oob_score=True,random_state=1)

classifier.fit(X_train,y_train)
```

Out[51]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features=12, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=1,
            oob_score=True, random_state=1, verbose=0, warm_start=False)
```

In [52]:
```
#Predicting the Test set results
y_pred= classifier.predict(X_test)
```

In [53]:
```
print((metrics.accuracy_score(y_test, y_pred)))
```

```
0.4606379368849678
```

Only 46%!!! :-(

Conclusions

- It’s a really complex game, I red max accuracy is 70%
- The home field advantage is an important factor.
- Draw is the less predictable of the outcomes
- Applying the methods to the data:

KNN	Logistic Regression	Random Forest
52%	51%	46%

Extensions

- Check at other level, for example go to Player level, how for example can affect the experience (age) of the player in the game outcome
- Use the betting companies data (matches) and match with our predictions
- Check the outcome with other data

Thank You

- Email: d.imperatori.bou@gmail.com
- GitHub: https://github.com/dimperatori