In [1]:

```
#You're gonna like this notebook if you want to brush up on SQL,get some tricks or want to revise certa
in procedures.
#You're gonna love this notebook if you like football.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sqlite3
%matplotlib inline


#The two research questions that I'll be concentrating on are:
# 1. How does Age of players correlate with Stamina, Reactions, Agility, Sprint Speed & Acceleration??
# 2. This is more Complex. I would like to find out which league is more unpredictable amongst
# EPL, Bundesliga, La Liga, Serie A and Ligue 1(Who watches anything else anyway?).
# For this I assign an Unpredictability score to every League. The steps to find this score is as follo
ws:
#    a) Form league standings from match scores. [3 points to a team that wins, 0 for those who lose &
1 a piece if match ends in draw.]
#    b) Find out the Top 5 teams and the Bottom 5 of each league in each season.
#    c) Find out the results of the matches between the Top 5 and the Bottom 5 in that particular seaso
n
#    d) If any of the bottom 5 teams beat any of the Top 5 teams at their own home ground give them 1 p
oint.
#       If they draw a match in their home ground give them 0.5 point.
#       If they manage to beat a Top 5 team away give them 1.25 points.
#       If they manage a draw with a Top 5 team away give them 0.75 points.
#       Add up all the scores for a season for each league and this is the Unpredictability of the leag
ue.
```

In [2]:

```
#connect connect connect let python meet SQL

connection = sqlite3.connect('database.sqlite')

#Every SQLite database has an SQLITE_MASTER table(read-only) that defines the schema for the database.
tables = pd.read_sql("""SELECT *
                        FROM sqlite_master
                        WHERE type='table';""",connection)
```

In [62]:

```
#deal witht the biggest baddest table first and life is easier thereafter
#Do not be afraid of SELECT *... try it out... its harmless.....

#Q: Why do you never ask SQL people to help you move your furniture?
#A: They sometimes drop the tables

match = pd.read_sql("""SELECT *
                        FROM Match;
                        """, connection)

print(match.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25979 entries, 0 to 25978
Columns: 115 entries, id to BSA
dtypes: float64(96), int64(9), object(10)
memory usage: 22.8+ MB
None
```

In [63]:

```
#they have an inbuilt for everything!!!!
match.isnull().sum(axis=0)
```

Out[63]:

```
id                       0
country_id               0
league_id                0
season                   0
stage                    0
date                     0
match_api_id             0
home_team_api_id         0
away_team_api_id         0
home_team_goal           0
away_team_goal           0
home_player_X1        1821
home_player_X2        1821
home_player_X3        1832
home_player_X4        1832
home_player_X5        1832
home_player_X6        1832
home_player_X7        1832
home_player_X8        1832
home_player_X9        1832
home_player_X10       1832
home_player_X11       1832
away_player_X1        1832
away_player_X2        1832
away_player_X3        1832
away_player_X4        1832
away_player_X5        1832
away_player_X6        1832
away_player_X7        1832
away_player_X8        1832
                     ...
B365H                 3387
B365D                 3387
B365A                 3387
BWH                   3404
BWD                   3404
BWA                   3404
IWH                   3459
IWD                   3459
IWA                   3459
LBH                   3423
LBD                   3423
LBA                   3423
PSH                  14811
PSD                  14811
PSA                  14811
WHH                   3408
WHD                   3408
WHA                   3408
SJH                   8882
SJD                   8882
SJA                   8882
VCH                   3411
VCD                   3411
VCA                   3411
GBH                  11817
GBD                  11817
GBA                  11817
BSH                  11818
BSD                  11818
BSA                  11818
Length: 115, dtype: int64
```

In [64]:

```
#drop 'em dead if they be NaN
match_imp = match.dropna(axis='columns')
print(match_imp.info())
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 25979 entries, 0 to 25978
Data columns (total 11 columns):
id                  25979 non-null int64
country_id          25979 non-null int64
league_id           25979 non-null int64
season              25979 non-null object
stage               25979 non-null int64
date                25979 non-null object
match_api_id        25979 non-null int64
home_team_api_id    25979 non-null int64
away_team_api_id    25979 non-null int64
home_team_goal      25979 non-null int64
away_team_goal      25979 non-null int64
dtypes: int64(9), object(2)
memory usage: 2.2+ MB
None
```

In [65]:

```
match_imp.duplicated().sum()
```

Out[65]:

```
0
```

In [66]:

```
#merge Match Information with league information
match_league = pd.read_sql("""SELECT m.country_id,lg.name,m.season,m.stage,m.date,m.match_api_id,m.home
_team_api_id,m.away_team_api_id,m.home_team_goal,m.away_team_goal
                              FROM match m
                              JOIN league lg
                              ON m.league_id = lg.id""",connection)
match_league.to_sql("match_league", connection, if_exists="replace")
print(match_league.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25979 entries, 0 to 25978
Data columns (total 10 columns):
country_id          25979 non-null int64
name                25979 non-null object
season              25979 non-null object
stage               25979 non-null int64
date                25979 non-null object
match_api_id        25979 non-null int64
home_team_api_id    25979 non-null int64
away_team_api_id    25979 non-null int64
home_team_goal      25979 non-null int64
away_team_goal      25979 non-null int64
dtypes: int64(7), object(3)
memory usage: 2.0+ MB
None
```

In [67]:

```
#All this work to create standings tables
match_league['date'] = pd.to_datetime(match_league['date'])
match_league['winner'] = np.where(match_league['home_team_goal']> match_league['away_team_goal'],match_
league['home_team_api_id'],match_league['away_team_api_id'])
match_league['winner'] = np.where(match_league['home_team_goal'] == match_league['away_team_goal'],9999
99,match_league['winner'])
match_league['draw1'] = np.where(match_league['home_team_goal'] == match_league['away_team_goal'],match
_league['home_team_api_id'],999999)
match_league['draw2'] = np.where(match_league['home_team_goal'] == match_league['away_team_goal'],match
_league['away_team_api_id'],999999)

match_league.to_sql("match_league", connection, if_exists="replace")
match_league.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25979 entries, 0 to 25978
Data columns (total 13 columns):
country_id          25979 non-null int64
name                25979 non-null object
season              25979 non-null object
stage               25979 non-null int64
date                25979 non-null datetime64[ns]
match_api_id        25979 non-null int64
home_team_api_id    25979 non-null int64
away_team_api_id    25979 non-null int64
home_team_goal      25979 non-null int64
away_team_goal      25979 non-null int64
winner              25979 non-null int64
draw1               25979 non-null int64
draw2               25979 non-null int64
dtypes: datetime64[ns](1), int64(10), object(2)
memory usage: 2.6+ MB
```

In [68]:

```
#check out your work of art
query = pd.read_sql("""SELECT *
                       FROM match_league
                       ;""",connection)
query.head()
```

Out[68]:

	index	country_id	name	season	stage	date	match_api_id	home_team_api_id	away_team_api_id	home_te
0	0	1	Belgium Jupiler League	2008/2009	1	2008-08-17 00:00:00	492473	9987	9993	1
1	1	1	Belgium Jupiler League	2008/2009	1	2008-08-16 00:00:00	492474	10000	9994	0
2	2	1	Belgium Jupiler League	2008/2009	1	2008-08-16 00:00:00	492475	9984	8635	0
3	3	1	Belgium Jupiler League	2008/2009	1	2008-08-17 00:00:00	492476	9991	9998	5
4	4	1	Belgium Jupiler League	2008/2009	1	2008-08-16 00:00:00	492477	7947	9985	1

In [69]:

```
#All ze stats of da Teams
home_draw = pd.read_sql("""SELECT name,season,draw1,count(draw1) AS dh
                           FROM match_league
                           WHERE draw1 != 999999
                           GROUP BY 1,2,3;""",connection)
away_draw = pd.read_sql("""SELECT name,season,draw2,count(draw2) AS da
                           FROM match_league m1
                           WHERE draw2 != 999999
                           GROUP BY 1,2,3;""",connection)
winner_t = pd.read_sql("""SELECT name,season,winner,count(winner) AS w
                           FROM match_league m1
                           WHERE winner != 999999
                           GROUP BY 1,2,3;""",connection)
home_draw.to_sql("home_draw", connection, if_exists="replace")
away_draw.to_sql("away_draw", connection, if_exists="replace")
winner_t.to_sql("winner_t", connection, if_exists="replace")
```

In [55]:

```
#Statz of da players
attribute = pd.read_sql("""SELECT pa.date,pl.birthday,pl.player_api_id,pl.player_name,pa.acceleration,p
a.sprint_speed,pa.stamina,pa.agility,pa.reactions,pa.preferred_foot
                           FROM player pl
                           JOIN player_Attributes pa
                           ON pl.player_api_id = pa.player_api_id;""",connection)
attribute['date'] = pd.to_datetime(attribute['date'])
attribute['birthday'] = pd.to_datetime(attribute['birthday'])

attribute.to_sql("attribute_imp",connection,if_exists="replace")
attribute.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 183978 entries, 0 to 183977
Data columns (total 10 columns):
date                  183978 non-null datetime64[ns]
birthday              183978 non-null datetime64[ns]
player_api_id         183978 non-null int64
player_name           183978 non-null object
acceleration          183142 non-null float64
sprint_speed          183142 non-null float64
stamina               183142 non-null float64
agility               181265 non-null float64
reactions             183142 non-null float64
preferred_foot        183142 non-null object
dtypes: datetime64[ns](2), float64(5), int64(1), object(2)
memory usage: 14.0+ MB
```

In [56]:

```
#Preparation of data is half the job
#Keeping most recent record of each player
attribute.drop_duplicates(subset=['player_api_id',],keep="first",inplace=True)
attribute.dropna(inplace=True)
attribute.to_sql("attribute_imp",connection,if_exists="replace")
```

In [72]:

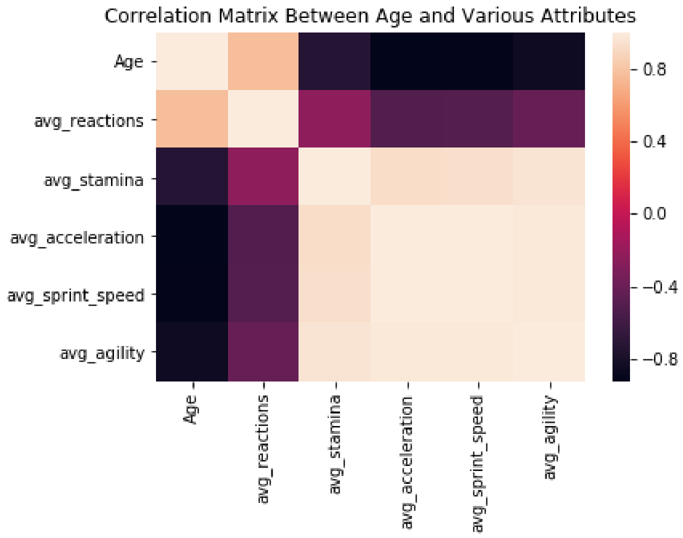
```
#Calculate the age of player

def num_years(start,curr):
    return(int((curr-start).days / 365.25))

query = pd.read_sql("""SELECT * FROM attribute_imp;""",connection)
query['date'] = pd.to_datetime(query['date'])
query['birthday'] = pd.to_datetime(query['birthday'])
#query['age'] = (query['date'].dt.year)-(query['birthday'].dt.year)
query['age'] = query.apply(lambda x: num_years(x['birthday'], x['date']), axis = 1)

query.to_sql("attribute_imp",connection,if_exists="replace")
```

In [73]:

```
import seaborn as sns
query1 = pd.read_sql("""  SELECT age AS Age,AVG(reactions) AS avg_reactions,AVG(stamina) AS avg_stamina
                          ,AVG(acceleration) AS avg_acceleration,AVG(sprint_speed) AS avg_sprint_speed
                          ,AVG(agility) AS avg_agility
                          FROM attribute_imp
                          GROUP BY 1
                          ORDER BY 1""",connection)
corr = query1.corr()
ax = sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values,)
ax.set_title("Correlation Matrix Between Age and Various Attributes")
```

Out[73]:

```
Text(0.5,1,'Correlation Matrix Between Age and Various Attributes')
```



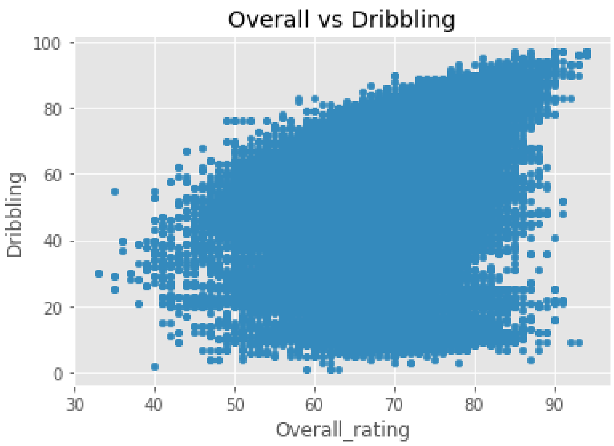
This Correlation matrix helps us find out how different attributes are linked to each other. We can clearly see that Age has a strong negative correlation with average acceleration,average agility, average stamina and average sprint speed of the players whereas age shows a weak positive correlation with the average reactions of the players.

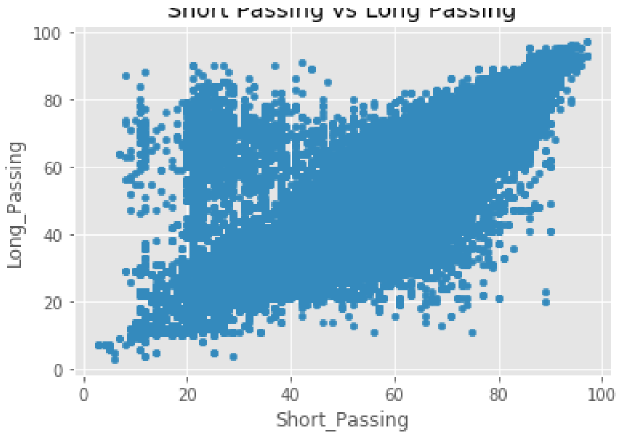
In [58]:

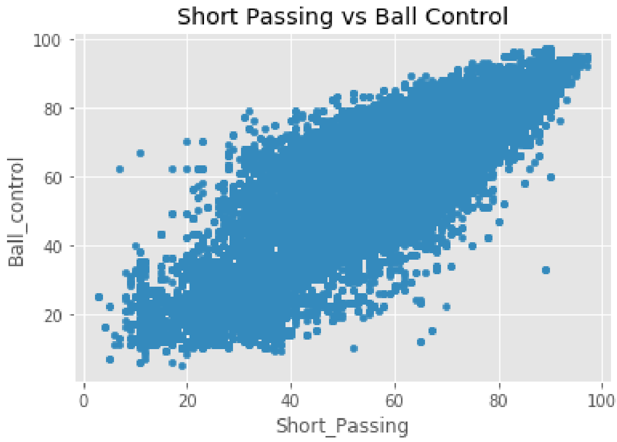
```
qu = pd.read_sql("""  SELECT overall_rating AS Overall_rating,potential AS Potential, dribbling AS Drib
bling,
                        short_passing as Short_Passing, long_passing AS Long_Passing, ball_control AS B
all_control
                        FROM Player_Attributes
                        ORDER BY 1""",connection)

ax = qu.plot(x="Overall_rating",y="Dribbling",kind="scatter")
ax.set_title("Overall vs Dribbling")

ax = qu.plot(x="Short_Passing",y="Long_Passing",kind="scatter")
ax.set_title("Short Passing vs Long Passing")

ax = qu.plot(x="Short_Passing",y="Ball_control",kind="scatter")
ax.set_title("Short Passing vs Ball Control")

ax = qu.plot(x="Overall_rating",y="Potential",kind="scatter")
ax.set_title("Overall vs Potential")
```

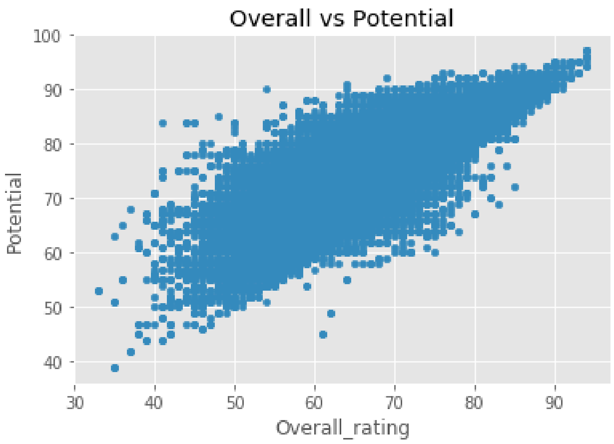
Out[58]:

```
Text(0.5,1,'Overall vs Potential')
```









In this section I have displayed various scatter plots between attributes that I thought might have links with each other. In terms of prediction analysis later we might infer that ball control and short passing form a good fit.

Players with higher overall ratings are not always the players who can dribble well and this is true as several high rated players are defenders and goalkeepers whose strong suit is not dribbling.

Surprisingly there are a lot of players who despite having high Long Passing scores have low Short Passing scores and this is a very interesting point

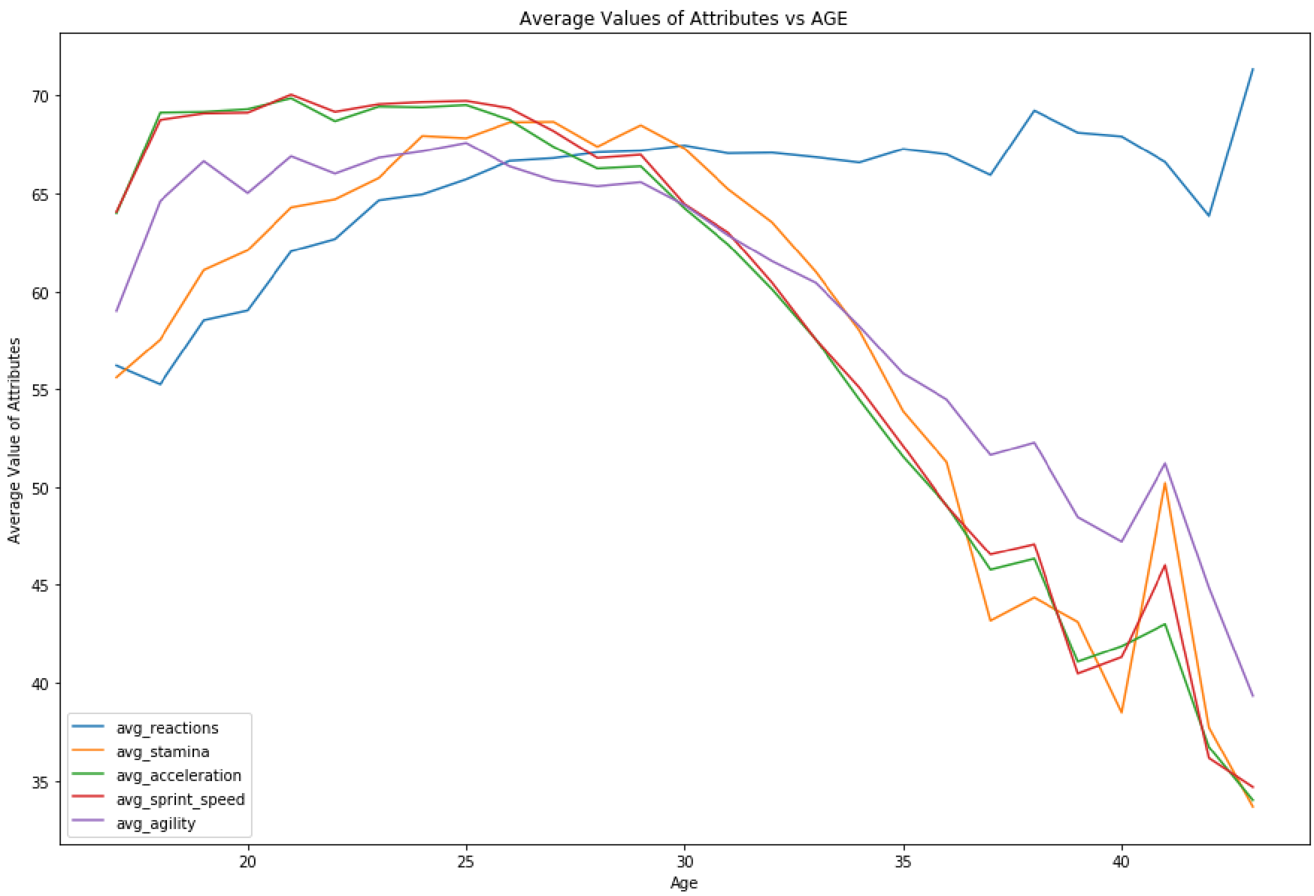
In [75]:

```
#This is where we plot
query1 = pd.read_sql("""  SELECT age AS Age,AVG(reactions) AS avg_reactions,AVG(stamina) AS avg_stamina
                          ,AVG(acceleration) AS avg_acceleration,AVG(sprint_speed) AS avg_sprint_speed
                          ,AVG(agility) AS avg_agility
                          FROM attribute_imp
                          GROUP BY 1
                          ORDER BY 1""",connection)
ax = query1.plot(x="Age", y=["avg_reactions", "avg_stamina", "avg_acceleration","avg_sprint_speed","avg
_agility"], kind="line",figsize=(15,10))
```

```
ax.set_ylabel("Average Value of Attributes")
ax.set_title("Average Values of Attributes vs AGE")
print("Correlation of Reactions with Age: ",query1['Age'].corr(query1['avg_reactions']))
print("Correlation of Stamina with Age: ",query1['Age'].corr(query1['avg_stamina']))
print("Correlation of Acceleration with Age: ",query1['Age'].corr(query1['avg_acceleration']))
print("Correlation of Sprint Speed with Age: ",query1['Age'].corr(query1['avg_sprint_speed']))
print("Correlation of Agility with Age: ",query1['Age'].corr(query1['avg_agility']))
```

```
C:\Users\dutta\Anaconda3\lib\site-packages\pandas\plotting\_core.py:1716: UserWarning: Pandas doesn't a
llow columns to be created via a new attribute name - see https://pandas.pydata.org/pandas-docs/stable/
indexing.html#attribute-access
  series.name = label
```

```
Correlation of Reactions with Age:  0.7675305570880387
Correlation of Stamina with Age:  -0.7260835589278937
Correlation of Acceleration with Age:  -0.9199811468752116
Correlation of Sprint Speed with Age:  -0.9104739545625228
Correlation of Agility with Age:  -0.849583392075151
```



In this graph we can clearly see that as the players get older the attributes which depend on the physical health of the player like Acceleration, Sprint Speed, Agility and Stamina decreases but reactions increases somewhat and I have explained more about this in the conclusions.

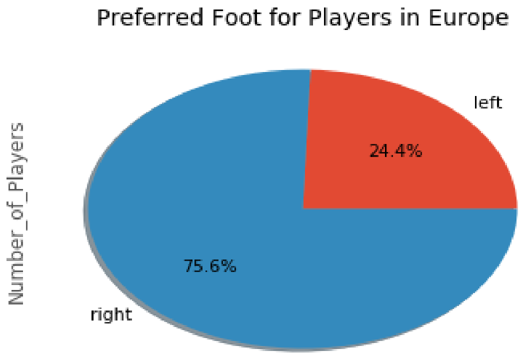
In [57]:

```
query = pd.read_sql("""SELECT preferred_foot AS Preferred_Foot,COUNT(*) as Number_of_Players
                       FROM attribute_imp
                       GROUP BY 1""",connection)




ax = query.plot(kind="pie",y='Number_of_Players', autopct='%1.1f%%',
 startangle=0, shadow=True, labels=query['Preferred_Foot'], legend = False, fontsize=11)
```

```
ax.set_title("Preferred Foot for Players in Europe")
query
```

Out[57]:

	Preferred_Foot	Number_of_Players
0	left	2583
1	right	7999



This is pretty clear as in only about 1 player in 4 is left footed. Majority of players are right footed.

In [76]:

```
#Forming league Tables
query = pd.read_sql("""SELECT hd.name,hd.season,hd.draw1 AS Team_id,hd.dh+ad.da+3*wi.w AS Points
                       FROM home_draw hd
                       JOIN away_draw ad
                       ON hd.name = ad.name AND hd.season=ad.season AND hd.draw1=ad.draw2
                       JOIN winner_t wi
                       ON hd.name = wi.name AND hd.season=wi.season AND hd.draw1 = wi.winner
                       WHERE hd.name LIKE "England Premier League" OR hd.name LIKE "France Ligue 1" OR
                       hd.name LIKE "Germany 1. Bundesliga" OR hd.name LIKE "Italy Serie A" OR
                       hd.name LIKE "Spain LIGA BBVA"
                       ORDER BY 1,2,4 DESC;""",connection)
query.to_sql("league_tables",connection,if_exists="replace")
```

In [77]:

```
#To find Top 5 and Bottom 5 of each league in each season
query = pd.read_sql("""SELECT * FROM league_tables;""",connection)
lar = (query.groupby(['name','season'],group_keys=False)).apply(lambda x: x.nlargest(5,'Points'))
sma=(query.groupby(['name','season'],group_keys=False)).apply(lambda x: x.nsmallest(5,'Points'))
```

In [78]:

```
#Evaluating head-to-head scores to find the Unpredictability of each league
query = pd.read_sql("""SELECT * FROM league_tables;""",connection)
lar = (query.groupby(['name','season'],group_keys=False)).apply(lambda x: x.nlargest(5,'Points'))

sma=(query.groupby(['name','season'],group_keys=False)).apply(lambda x: x.nsmallest(5,'Points'))

query1 = pd.read_sql("""SELECT * FROM match_league
                        WHERE name IN ("France Ligue 1","England Premier League","Spain LIGA BBVA","Ger
many 1. Bundesliga","Italy Serie A")
                        ORDER BY name,date;""",connection)
l=0

c=[0]*40
ss = []
for k in range(0,200,5):
```

```
    for i in range(k,k+5):
        for j in range(k,k+5):
            sid = sma.iloc[i,3] #Team_id of one of the Bottom 5
            lid = lar.iloc[j,3] #Team_id of one of the Top 5
            s=sma.iloc[i,2]      #Season for which we are evaluating
            ss.append(s)
            #When bottom 5 teams plays the Top 5 teams at their home
            a = query1.loc[query1.home_team_api_id == sid] #Filtering by home team
            b = a.loc[(query1.away_team_api_id == lid)]     #Filtering by away team
            d = b.loc[(query1.season == s)]                 #Filtering by season

            if((not d.empty)):
                if((d.iloc[0,11]==sid)):
                    c[l] = c[l] + 1
                elif((d.iloc[0,11]==999999)):
                    c[l] = c[l] + 0.5

            #When bottom 5 teams plays the Top 5 teams away
            a = query1.loc[query1.home_team_api_id == lid]
            b = a.loc[(query1.away_team_api_id == sid)]
            d = b.loc[(query1.season == s)]
            if((not d.empty)):
                if((d.iloc[0,11]==sid)):
                    c[l] = c[l] + 1.25
                elif((d.iloc[0,11]==999999)):
                    c[l] = c[l] + 0.5


    l=l+1
```

In [79]:

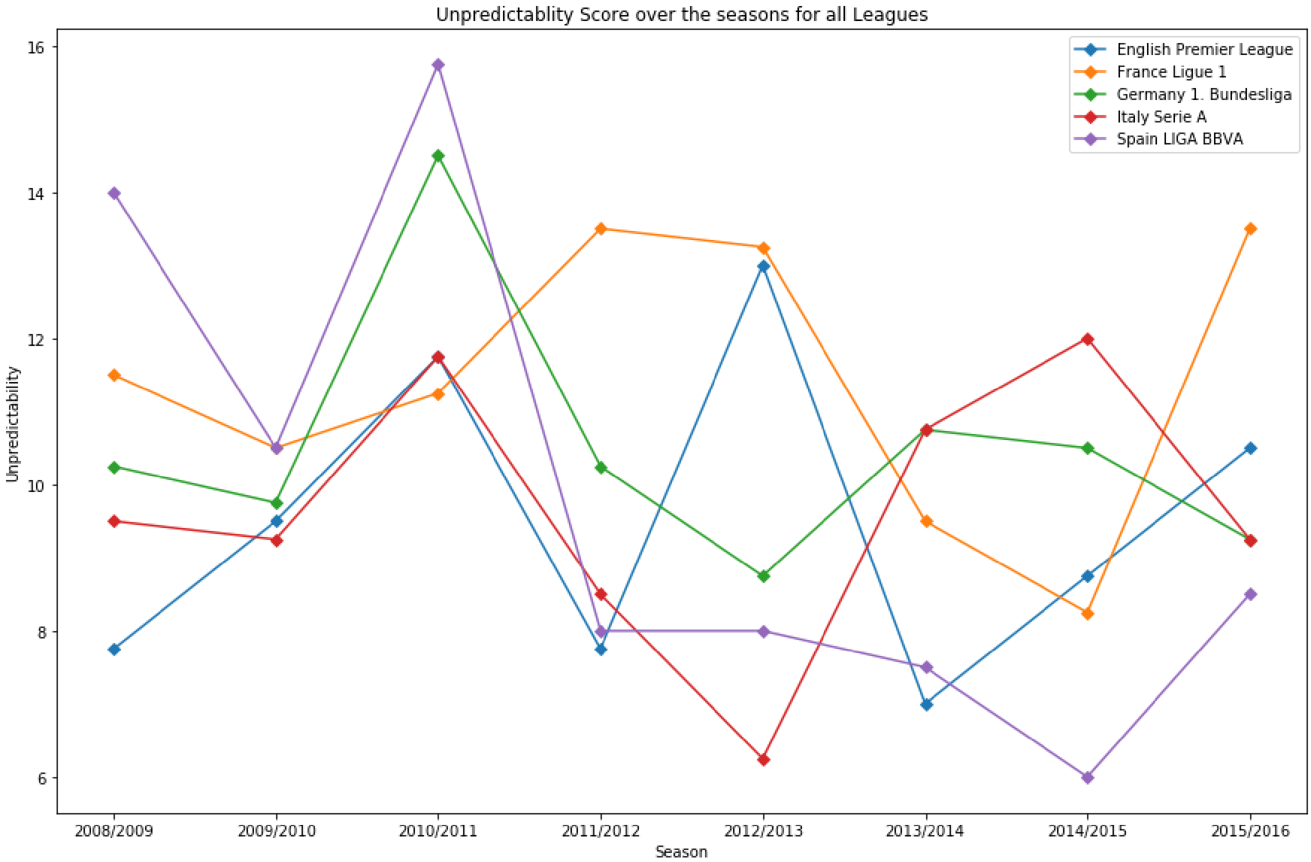
```
from collections import OrderedDict
a=list(OrderedDict.fromkeys(ss))
df = {'English Premier League':pd.Series(data=c[0:8],index=a),
      'France Ligue 1':pd.Series(data=c[8:16],index=a),
      'Germany 1. Bundesliga':pd.Series(data=c[16:24],index=a),
      'Italy Serie A':pd.Series(data=c[24:32],index=a),
      'Spain LIGA BBVA':pd.Series(data=c[32:40],index=a)}
df=pd.DataFrame(df)
ax = df.plot(figsize=(15,10),marker='D')
ax.set_xlabel("Season")
ax.set_ylabel("Unpredictability")
ax.set_title("Unpredictablity Score over the seasons for all Leagues")
x= [0, 1, 2, 3, 4, 5, 6, 7]
labels =['2008/2009','2009/2010','2010/2011','2011/2012','2012/2013','2013/2014','2014/2015','2015/2016
']
plt.xticks(x,labels)
plt.subplots_adjust(bottom=0.15)
print("Average Unpredictability: \n",df.mean(axis=0))
df
```

```
Average Unpredictability: 
 English Premier League     9.50000
France Ligue 1            11.40625
Germany 1. Bundesliga     10.50000
Italy Serie A              9.65625
Spain LIGA BBVA            9.78125
dtype: float64
```

Out[79]:

	English Premier League	France Ligue 1	Germany 1. Bundesliga	Italy Serie A	Spain LIGA BBVA
2008/2009	7.75	11.50	10.25	9.50	14.00
2009/2010	9.50	10.50	9.75	9.25	10.50
2010/2011	11.75	11.25	14.50	11.75	15.75
2011/2012	7.75	13.50	10.25	8.50	8.00
2012/2013	13.00	13.25	8.75	6.25	8.00

	English Premier League	France Ligue 1	Germany 1. Bundesliga	Italy Serie A	Spain LIGA BBVA
2013/2014	7.00	9.50	10.75	10.75	7.50
2014/2015	8.75	8.25	10.50	12.00	6.00
2015/2016	10.50	13.50	9.25	9.25	8.50



The Unpredictability Score of various leagues over the years gives us an insight of what each league is like. I have explained some of the specific season details in the conclusion.

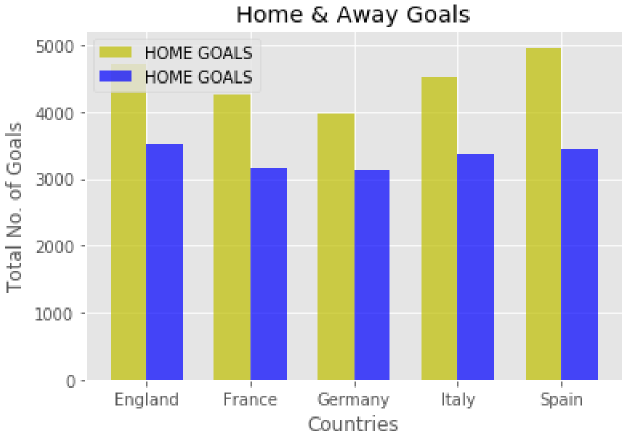
In [59]:

```
query = pd.read_sql("""SELECT name,SUM(home_team_goal) as HOME,SUM(away_team_goal) AS AWAY
                        FROM match_league
                        WHERE name IN ("France Ligue 1","England Premier League","Spain LIGA BBVA","Ger
many 1. Bundesliga","Italy Serie A")
                        GROUP BY 1""",connection)

ind = np.arange(5)
width = 0.35

hm = plt.bar(ind, query['HOME'],width,color='y',alpha=0.7,label='HOME GOALS')
aw = plt.bar(ind+width, query['AWAY'],width,color='b',alpha=0.7,label='HOME GOALS')
plt.ylabel('Total No. of Goals')
plt.xlabel('Countries')
plt.title('Home & Away Goals')
locations = ind+width / 2
labels = ["England","France","Germany","Italy","Spain"]
plt.xticks(locations,labels)
plt.legend()
query
```

Out[59]:

	name	HOME	AWAY
0	England Premier League	4715	3525
1	France Ligue 1	4265	3162

	name	HOME	AWAY
2	Germany 1. Bundesliga	3982	3121
3	Italy Serie A	4528	3367
4	Spain LIGA BBVA	4959	3453



No doubts here, home advantage is a very big part of football. Very interestingly even though La Liga takes the cake in terms of most number of goals score EPL still has more away goals scored. Also important to note here that even though the German League has significantly lower number of matches(2448 for 8 seasons whereas others have 3040) it still rakes in a lot of goals and has the highest average goals per game amongst all.

Conclusions

A basic analysis of various available data on European Soccer matches was done here. Some takeaways are:

1. Contrary to what you may believe Ligue 1 seems most unpredictable i.e. a bottom 5 team gets a favourable outcome against a Top 5 team most often in this league.
2. La liga had the most unpredictable league and that was 2010-11 but also the most predictable league with lowest unpredictability score in 2014-15
3. Germany has a general high unpredictableness but it may be due to the reason that I have considered the Top & Bottom 5 even though this league has only 18 teams each season. Having said that,2010-11 was a crazy season in Bundesliga. Dortmund was champion, Bayern Munich came 3rd, Schalke, VfL Wolfsburg, Borussia Mönchengladbach, Eintracht Frankfurt were amongst the lowest ranked teams. Suprisingly Schalke went on to win the DFB Pokal and competed in the Europa League despite being so lowly ranked.
4. In terms of Player Data, there is a strong Negative correlation of Age with attributes like Sprint Speed, Acceleration, Agility and Stamina whereas Reactions show a strong Positive correlation.
5. Generally all these attributes are maximum at 26-30 which is widely known as the football peak ages.
6. In terms of various attribute comparison we can see that the short pass and ball control attributes seem to have a nice fit which makes sense(think Iniesta)
7. Obviously the number of home goals scored are way more than the number of away goals scored. I pity away teams playing in grounds like Signal Iduna Park,Old Trafford etc absolute crazy atmosphere

Limitations

1. pd.read_sql() in Python runs SQLite syntax only and thus I was not able to apply window functions which would have made life simpler at some parts.
2. There are some outliers in data but they are important and thus cannot be removed. From the reactions vs age graph we see that reactions suddenly increase after age of 42 which seems odd. On further inspection I found 3-4 goalkeeprs(e.g David James) which provided this sudden rise.