ganesh_european_soccer_matches

August 2, 2019

1 Project: Investigate Kaggle's European Soccer Database

1.1 Table of Contents

Introduction

As a soccer fan and season pass holder for the Netherlands' best football club Ajax, it was not hard to choose between the datasets provided for this project. Picking the most interesting research questions, however, was much more difficult: with dozens of variables for matches, teams and individual players, there were literally hundreds of questions to answer. After much thought, I decided to extract two datasets from the SQLite database (the SQL statements that I used are described in the next cell):

- Matches (league name, date, home team, away team and the goals scored for the respective teams)
- Players (a combination of the tables 'player' and 'player_attributes')

For these datasets, I will try to answer the following research questions: 1. Is there an advantage for teams to playing in their own stadium (home advantage)? 2. How did the Dutch football club Ajax perform over the years, compared to its main competitors PSV and Feyenoord? 3. Are there more professional football players born in the first months of the year than in the last months of the year (relative age effect)? 4. Are there player characteristics that correspond to higher overall rating in the EA Sports FIFA video game?

The European Soccer database was stored in a SQLite database. I used the software 'DB Browser for SQLite' and the following SQL statements to extract the datasets I needed:

Matches

```
SELECT l.name AS league_name, m.season, m.date, t1.team_long_name as home_team, t2.team_long_nam
FROM Match m
JOIN League l
ON m.league_id = l.id
INNER JOIN Team as t1
ON m.home_team_api_id = t1.team_api_id
```

```
INNER JOIN Team as t2
ON m.away_team_api_id = t2.team_api_id;
```

Players

```
SELECT *
FROM Player_Attributes pa
JOIN Player p
ON p.player_api_id = pa.player_api_id;
```

```
In [ ]: #Import statements for libraries needed for this project
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from datetime import date
        import seaborn as sns
        import calendar
        % matplotlib inline
```

```
In [ ]: #Set display options in order the inspect the data in the extensive datasets properly
        pd.set_option('display.height', 1000)
        pd.set_option('display.max_rows', 100)
        pd.set_option('display.max_columns', 100)
        pd.set_option('display.width', 1000)
```

Data Wrangling: Matches

```
In [ ]: #Import dataset
        matches = pd.read_csv('european_soccer_matches.csv')
```

1.1.1 Notes on data cleaning and wrangling: Matches

- There were no duplicate rows in the Matches dataframe, so no action had to be taken.
- In the 25979 rows in the dataframe, no null values were found.
- The dates in column 'date' were strings when imported, I converted this column to datetime format
- To be able to make statements about the presumed advantage for teams playing in their home stadium, I created a new column for net score column.
- For the last research question about my favourite team Ajax, I created a filtered dataframe. I added a new column to this dataframe based on the net score in which the number of points for Ajax were calculated (in football, you will get 3 points for a win, 1 point for a draw and no points for a loss). I did the same for Ajax' competitors PSV and Feyenoord.

```
In [ ]: #Find number of duplicate rows
        sum(matches.duplicated())
```

```
In [ ]: #Check if there are any null values in the dataframe
        matches.info()
```

```
In [ ]: #Column 'date' is currently a string, convert to datetime format
        matches['date'] = pd.to_datetime(matches['date'])

In [ ]: #Create new column with day of week
        matches['day_of_week'] = matches['date'].dt.weekday_name

In [ ]: #Calculate the net score for each game. This new column will be used to determine if it
        net_score = matches['home_team_goal'] - matches['away_team_goal']
        matches['net_score'] = net_score

In [ ]: #Create separate dataframe with Ajax matches only and do the same for the main competito
        ajax = matches[(matches.home_team == 'Ajax') | (matches.away_team == 'Ajax')]
        ajax = ajax.drop(columns=['league_name'])
        psv = matches[(matches.home_team == 'PSV') | (matches.away_team == 'PSV')]
        psv = psv.drop(columns=['league_name'])
        feyenoord = matches[(matches.home_team == 'Feyenoord') | (matches.away_team == 'Feyenoor
        feyenoord = feyenoord.drop(columns=['league_name'])

In [ ]: #Write function to calculate the number of points for the team that is being investigate
        #are awarded for a win, 1 point for a draw and 0 points for a loss.
        def points(row, team):
            if (row['net_score'] > 0) and (row['home_team'] == team):
                return 3
            elif (row['net_score'] < 0) and (row['away_team'] == team):
                return 3
            elif (row['net_score'] > 0) and (row['away_team'] == team):
                return 0
            elif (row['net_score'] < 0) and (row['home_team'] == team):
                return 0
            else:
                return 1

In [ ]: #Create new column in the team dataframes that contains the number of points earned per
        ajax['points_ajax'] = ajax.apply(lambda row: points(row, 'Ajax'), axis=1)
        psv['points_psv'] = psv.apply(lambda row: points(row, 'PSV'), axis=1)
        feyenoord['points_feyenoord'] = feyenoord.apply(lambda row: points(row, 'Feyenoord'), ax

In [ ]: #Write function to calculate the number of goals for the team that is being investigated
        def goals(row, team):
            if row['home_team'] == team:
                return row['home_team_goal']
            else:
                return row['away_team_goal']

In [ ]: #Create new column in the team dataframes that contains the number of goals scored per m
        ajax['goals_ajax'] = ajax.apply(lambda row: goals(row, 'Ajax'), axis=1)
        psv['goals_psv'] = psv.apply(lambda row: goals(row, 'PSV'), axis=1)
        feyenoord['goals_feyenoord'] = feyenoord.apply(lambda row: goals(row, 'Feyenoord'), axis
```

Exploratory Data Analysis: Matches

1.1.2 Research Question 1: Is there an advantage for teams to playing in their own stadium?

```
In [ ]: #Explore contents of dataframe for reference
        matches.head()

In [ ]: #Descriptive statistics for the matches dataset
        matches.describe()
```

If a home advantage exists, one would expect the mean net score to be zero, which means that on average, the same number of goals are scored by the home team and the away team. The mean net score for this dataset, however, is 0.38. To see if this result is statistically significant, I ran a one sample t-test to compare the mean net score to the mean net score to be expected without home advantage (0).

The formula for the one sample t-test is:

$t = \frac{m-\mu}{s/\sqrt{n}}$

Using the mean and the standard deviation for net_score from the table above:

$t = \frac{0.383656-0}{1.782403/\sqrt{25979}} = 34.69$

Looking up this value in a (T-Distribution table) learns that the p-value for this score is lower than the lowest p-value in the table. By conventional criteria, this difference in net score is considered to be statistically significant, so we can say that there is indeed an advantage for teams playing in their home stadium.

Although the phenomenon of home advantage is widely known in many sports, there is still much debate about the underlying reasons for it. Is it the cheering crowd, the familiarity of the stadium, the fact that players do not have to travel and sleep in their own bed, or even the referee who may be biased?

One researcher even suggested that the home advantage for teams in the German Bundesliga dissapears when a match is played in the middle of the week. Indeed, the average net score is lower when playing matches on Tuesday, Wednesday or Thursday but the home advantage does not dissapear completely, at least not for matches for the German Bundesliga period 2008-2016.

```
In [ ]: matches[(matches.league_name == 'Germany 1. Bundesliga') & ((matches.day_of_week == 'Tue
```

The extent to which the home advantage occurs varies between the leagues in the dataset. It would be plausible to expect that the home advantage is greater when the distances between home and away clubs are long. However, looking at the chart below, this seems not the case. Although the lowest mean net score is seen in the relative small country of Switzerland (which would fit the hypothesis), the highest mean net score is measured in Belgium, the smallest country in the dataset.

```
In [ ]: #Create bar chart for the net score per league
        mean_net_score = matches.pivot_table(index='league_name', values='net_score', aggfunc=np
        ax = mean_net_score.sort_values(by='net_score', ascending=False).plot.barh(x=mean_net_sc
        ax.set_xlabel("Mean net score")
        ax.set_ylabel("League name")
        ax.grid(False)
        plt.title('Mean net score per league', fontsize=14)
```

1.1.3 Research Question 2: How did the Dutch football club Ajax perform over the years, compared to its main competitors PSV and Feyenoord?

In the Netherlands, the highest football league Eredivisie is historically dominated by three clubs: Ajax from Amsterdam, PSV from Eindhoven and Feyenoord from Rotterdam. How did Ajax perform in terms of the number of points earned per season and the number of goals scored compared to its competitors?

```
In [ ]: #Create a combined dataframe of pivot tables, in order to compare the number of points e
        points_a = ajax.pivot_table(index='season', values=['points_ajax'],aggfunc=np.sum)
        points_p = psv.pivot_table(index='season', values=['points_psv'],aggfunc=np.sum)
        points_f = feyenoord.pivot_table(index='season', values=['points_feyenoord'],aggfunc=np.
        dfs = [points_a, points_p, points_f]
        combined_points = pd.concat(dfs, axis=1)
```

```
In [ ]: #Create grouped bar chart for the total number of points per season, per team
        labels = ['Ajax', 'PSV', 'Feyenoord']
        ax = combined_points.plot(kind='bar',figsize=(15,5), rot=0, colormap='RdYlGn')
        ax.legend(labels)
        ax.grid(False)
        plt.xlabel('Season', fontsize=12)
        plt.ylabel('Points', fontsize=12)
        plt.title('Big Three: Points per season', fontsize=14)
```

Over the the seasons between 2008 and 2016, Ajax was the overall winner with regard of the number of points earned, with PSV as runner up. However, in the first two seasons in the dataset there was a for the Dutch Eredivisie very special situation to have a champion that was not from the Big Three (see table below). The champion with the most points was PSV in season 2014/2015 (88 points), whereas Ajax only got 71 points and still won the league in 2013/2014. For Feyenoord, 2018-2016 were not the best years, with an all-time low in season 2010/2011 when the club ended up 10th place in the final rankings with only 44 points.

Season	Champion
2008/2009	AZ
2009/2010	FC Twente
2010/2011	Ajax
2011/2012	Ajax
2012/2013	Ajax
2013/2014	Ajax
2014/2015	PSV
2015/2016	PSV

```
In [ ]: #Create a combined dataframe of pivot tables, in order to compare the number of goals sc
        goals_a = ajax.pivot_table(index='season', values=['goals_ajax'],aggfunc=np.sum)
        goals_p = psv.pivot_table(index='season', values=['goals_psv'],aggfunc=np.sum)
        goals_f = feyenoord.pivot_table(index='season', values=['goals_feyenoord'],aggfunc=np.su
        dfs_g = [goals_a, goals_p, goals_f]
        combined_goals = pd.concat(dfs_g, axis=1)
```

```
In [ ]: #Create grouped bar chart for the total number of goals per season, per team
        ax = combined_goals.plot(kind='bar',figsize=(15,5), rot=0, colormap='RdYlGn')
        ax.legend(labels, loc='upper left')
        ax.grid(False)
        plt.xlabel('Season', fontsize=12)
        plt.ylabel('Goals', fontsize=12)
        plt.title('Big Three: Goals per season', fontsize=14)
```

The chart shows that the club that scores the most goals is not the champion by definition. For three out of four seasons (2010/2011 to 2013/2014) in which Ajax became champion, another team (PSV twice, Feyenoord once) scored more goals. As a result, in the period covered by this dataset, Ajax is the club that became champion with the fewest goals (69, in season 2013/2014). True to their famous offensive playing style, Ajax is also responsible for the most goals in one season in this dataset: 106 goals in season 2009/2010 (the season in which the title was grabbed by AZ Alkmaar).

Data Wrangling: Players

```
In [ ]: #load player data
        players = pd.read_csv('player_names_attributes.csv', index_col='id')
```

1.1.4 Notes on data cleaning and wrangling: Players

- There was one duplicate row in the players dataframe, I decided to drop that one row.

- In the players dataset that originally contained more than 183.000 rows, there were 3230 rows that contained null values. Because I wanted to make statements about larger groups of players (as opposed to making sure that every individual player is present in the dataset) I decided to drop all rows with null values.
- As weight is measured in kilograms in the Netherlands, I converted the column "weight" to kilograms instead of lbs.
- I added a column 'BMI' (Body Mass Index) to have a little more information about the ratio between weight and height.
- To be able to compare the ages of players, I first converted every row in the column 'birthday' to a datetime object and created a new column "age" from that. I borrowed the code for calculating the age at time of measurement here and here.

```
In [ ]: #To make the dataframe a bit more manageable, columns that will not be used are dropped
        players = players.drop(columns=['player_fifa_api_id', 'player_api_id', 'id.1', 'player_a
```

```
In [ ]: #Count number of duplicate rows and drop duplicates
        sum(players.duplicated())
        players.drop_duplicates(inplace=True)
```

```
In [ ]: #Check if there are any null values in the dataframe
        players.info()
```

```
In [ ]: #Find number of rows with missing values per column
        players.isnull().sum()
        #Drop rows with missing values
        players.dropna(axis=0, how='any', inplace=True)
        #Running players.info() again learns that every column now has the same number (180354)
```

```
In [ ]: #Create new column 'weight_kg' that contains the player's weight in kilograms
        weight_kg = players['weight'] * 0.45359237
        players['weight_kg'] = weight_kg

In [ ]: #Calculate BMI with the following formula: weight in kilograms/height in meters squared
        bmi = players['weight_kg']/(players['height']/100)**2
        players['bmi'] = bmi

In [ ]: #Function to calculate the age of the player on the rating date
        players['birthday'] = pd.to_datetime(players['birthday'])
        players['date'] = pd.to_datetime(players['date'])

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

        players['age'] = players.apply(lambda x: num_years(x['birthday'], x['date']), axis=1)

In [ ]: players['age'].hist(bins=20)
```

When checking the distribution for the newly created column 'age', apparently there are players younger than 15 or even 10 years at the time of ranking. As this is not possible for professional football players, I ran a query on the players younger than 15 (see cell below). It seems that there is a problem with one of the ranking dates (February 22, 2007). Therefore, when I want to investigate age later in this report, I will leave out the measurements of February 2007. I do not prefer to drop those rows because there are no problems with the rest of the data.

```
In [ ]: players.query('age < 15')

In [ ]: #Create new column with birth month (research question 3)
        players['birth_month'] = players['birthday'].dt.month

In [ ]: #Create dataframe with unique players to count every birthday just once
        unique_players = players.drop_duplicates(['player_name'])
        unique_players['birth_month'] = unique_players['birth_month'].apply(lambda x: calendar.m
```

Exploratory Data Analysis: Players

1.1.5 Research Question 3: Are there more professional football players born in the first months of the year than in the last months of the year?

```
In [ ]: #Create barplot with the proportion of players born in a specific month
        counts = unique_players['birth_month'].value_counts(normalize=True)
        months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep","Oct", "Nov", "D
        mean = 1/12

        sns.set_style("whitegrid")
        fig, ax = plt.subplots(figsize=(10,5))
        sns.barplot(x=months, y=counts, palette='GnBu_d')
        plt.ylabel("Proportion of players")
        plt.title('Birth month of professional football players')
        plt.axhline(mean, color='r', linestyle='--', label='mean')
```

On Wikipedia, the relative age effect is defined as follows:

> The term relative age effect (RAE) is used to describe a bias, evident in the upper echelons of youth sport and academia, where participation is higher amongst those born early in the relevant selection period (and correspondingly lower amongst those born late in the selection period) than would be expected from the normalised distribution of live births.

In Europe, the selection year for football starts at January 1st. Therefore, if a relative age effect is present, an overrepresentation of birth months in the first quarter of the year is expected. Looking at the table above, this trend is indeed visible. Unfortunately, at this point during my study, I cannot determine whether this difference is significant or if this observed distribution is due to chance. Nevertheless, as many researchers have demonstrated the existence of the relative age effect both in sports and education, the trend in the chart above does seem to indicate that an age effect exists for the professional football players in the dataset.

1.1.6 Research Question 4: Are there player characteristics that correspond to higher overall rating in the EA Sports FIFA video game?

```
In [ ]: players.describe()
```

In general, you can quickly see that there is a lot of variation in the mean and standard deviation across the various FIFA attributes. But the dry numbers become much more interesting (at least to me) when you realize that behind every extreme value, there are names of real players. For example, the world of difference between the top player with overall ranking of 94 (FC Barcelona's Lionel Messi) and the one with the lowest ranking (the Italian player Francesco Della Rocco who, fortunately for him, was very young when his first ranking was set and later reached an overall rating of 68 (**source**). The best header (the rather unknown Nicola Zigic), the most agile player (again, Lionel Messi amongst others), the most aggressive player (not surprisingly: Gennaro Gattuso, amongst others) and the one who scores penalties best (Rickie Lambert)... it is tempting to connect all attributes to names.

There is also a lot of variation in age, weight and height. For age, there is probably an error in the data because the minimum age is 7 in the dataset. I will deal with this problem before making statements about age.

To conclude, the tallest player in the set is Kristof van Hout, a Belgian goalkeeper who measures 2.08 meters and is also the heaviest player. The smallest player is Juan Quero, a Spanish midfielder, who is -with his 53 kilograms- also the lightest player in the set.

```
In [ ]: #Find names for extreme values from the descriptive statistics table
        players.query('overall_rating == 94')
        players.query('overall_rating == 33')
        players.query('heading_accuracy == 98')
        players.query('aggression == 97')
        players.query('penalties == 96')
        players.query('age == 43')
        players.query('height >= 208')
        players.query('height <= 158')
        players.query('weight_kg >= 110')
        players.query('weight_kg < 53.1')
```

```
In [ ]: #Create scatterplot for overall rating and some FIFA variables: potential, reactions and
        fig = plt.figure(figsize=(18,5))
        ax1 = fig.add_subplot(1,3,1)
        ax1.scatter(players['potential'], players['overall_rating'])
        plt.xlabel('Potential')
        plt.ylabel('Overall Rating')
        ax2 = fig.add_subplot(1,3,2)
        ax2.scatter(players['reactions'], players['overall_rating'])
        plt.xlabel('Reactions')
        plt.title('Scatterplots for Overall Rating and Potential, Reactions and Dribbling', font
        ax3 = fig.add_subplot(1,3,3)
        ax3.scatter(players['dribbling'], players['overall_rating'])
        plt.xlabel('Dribbling')
        plt.ylabel('Overall rating')

        #Calculate Pearson correlation coefficient for overall rating and potential/reactions/dr
        print('Correlation between overall rating and potential, reactions and dribbling respect
        print(players['potential'].corr(players['overall_rating']))
        print(players['reactions'].corr(players['overall_rating']))
        print(players['dribbling'].corr(players['overall_rating']))
```

To determine which player characteristics correspond to a higher overall FIFA rating, I first created scatterplots for every FIFA attribute and see if they are positively correlated (ie higher ratings on a specific attribute tend to go together with high overall ratings). To improve readability of this report, there are just three scatterplots in the cell above (the same code was used for every scatterplot).

Although I expected a strong correlation between the overall rating and at least some of the more general player characteristics like strength or stamina, this is not always the case. In fact, only potential (which makes sense for high rated players) and reactions did show a very strong positive relationship with overall rating.

Overview of Pearson correlation coefficient between overall rating and: * potential: 0.77 * crossing: 0.36 * finishing: 0.33 * heading accuracy: 0.31 * short_passing: 0.46 * volleys: 0.36 * dribbling: 0.35 * curve: 0.36 * free_kick_accuracy: 0.35 * long_passing: 0.44 * ball_control: 0.44 * acceleration: 0.26 * sprint_speed: 0.25 * agility: 0.24 * reactions: 0.77 * balance: 0.16 * shot_power: 0.43 * jumping: 0.26 * stamina: 0.33 * strength: 0.32 * long_shots: 0.39 * aggression: 0.32 * interceptions: 0.25 * positioning: 0.37 * vision: 0.43 * penalties: 0.39 * marking: 0.13 * standing_tackle: 0.13 * sliding_tackle: 0.13

I added these results to a dictionary and used this dictionary to create a bar chart.

```
In [ ]: #Create bar chart for skills that show the highest correlation with overall rating
        skills = ['potential', 'reactions', 'short passing', 'long passing', 'ball control', 'sh
                  'long shots', 'penalties', 'positioning', 'volleys', 'crossing', 'curve', 'dri
                  'finishing', 'stamina', 'strength', 'aggression', 'heading accuracy', 'jumping
                  'sprint speed', 'interceptions', 'agility', 'balance', 'marking', 'standing ta
        r =  [0.77, 0.77, 0.46, 0.44, 0.44, 0.43, 0.43, 0.39, 0.39, 0.37, 0.36, 0.36, 0.36, 0.35
              0.31, 0.26, 0.26, 0.25, 0.25, 0.24, 0.16, 0.13, 0.13, 0.13]

        fig, ax = plt.subplots(figsize=(12,12))
```

```
sns.set_style("dark")
sns.barplot(x=skills, y=r, palette='Blues_r')
sns.despine(left=True, bottom=True)
plt.xlabel('Skills', fontsize=14)
plt.ylabel('Pearson correlaton coefficient', fontsize=14)
plt.title('Correlation overall rating and field player skills', fontsize=14)
ax.set_xticklabels(skills, rotation=75, fontsize=14)
```

The remaining characteristics are all specific for goalkeepers so I would like to correlate these characteristics only to the goalkeepers in the dataset. Unfortunately, there is no single identifier to separate the goalies from the field players. I would have expected the columns with the goalkeeper characteristics to be empty for the field players but unfortunately that is not the case. At sofifa.com, the site that contains all the player ratings for EA Sports game FIFA 18, it appears that field players do not have a rating higher than 40 for the GK reflexes attitude, which means that if we filter the player data on gk_reflexes we should only have goalkeepers. To be on the safe side, I chose 50 as a threshold for goalkeeping reflexes.

Looking at the list of unique names in the goalkeepers dataset (see cell below), there are a lot of familiar (goalkeeper) names. Googling some random unknown names learns that can safely assume this indeed is a list of goalkeepers only.

```
In [ ]: gk = players[players['gk_reflexes'] >= 50]
        gk['player_name'].unique()
```

Next, we can see which goalkeeper characteristics correlate most strongly with the overall rating of the goalkeeper in the same way as the field players earlier.

Overview of Pearson correlation coefficient between overall rating and: * Diving: 0.91 * Handling: 0.89 * Kicking: 0.70 * Positioning: 0.90 * Reflexes: 0.91

As expected, all attributes that are specific to goalkeepers are strongly correlated with the overall rating of the goalkeeper. Only kicking skills seem to go together with high overall ratings a little less often, but there is still a strong positive relation with the overall rating. Because there were only a few characteristics that contributed to a higher overall rating for the field players, I decided not to analyze them all for the goalkeepers as well but try a different angle first.

Next to looking into the attributes that go together with a higher overall rating, you can also create two groups: one with a high overall rating (top 25%) and one with a low rating (bottom 25%), and see if there are any differences between the two groups. I created an additional filter to leave out the goalkeepers, as the skill scores for goalkeepers and field players are too different.

```
In [ ]: #Create two subgroups from the players dataframe, based on the first and third quartile
        #Exclude goalkeepers
        top_players = players[(players['overall_rating'] > 73) & (players['gk_reflexes'] <= 40)]
        bottom_players = players[(players['overall_rating'] < 64) & (players['gk_reflexes'] <= 4
```

```
In [ ]: #Descriptive statistics for the top players
        top_players.describe()
```

```
In [ ]: #Descriptive statistics for the bottom players
        bottom_players.describe()
```

Comparing the means of the numerical variables gives a good understanding of the skills you obviously need as a top player: the largest differences (>19 points in mean rating between top and bottom players) can be seen on the attributes long shots and shot power, followed by (>18 points difference in mean rating) ball control, reactions, positioning, vision, volleys and curve.

To see the shape of the distributions, I created two visuals below: one for an attribute that is typical for top players (finishing) and one for which the mean rating for top players is almost the same as for bottom players (sliding tackle).

The distribution for finishing is as I expected (bottom players to the left with lower rating, top players on the right), but the distribution of the sliding tackles appears to be bimodal for the top players. It is a good example of a situation where just comparing the mean can be deceptive because when looking at the median for both groups, the top players once again come out as winners when it comes to their sliding tackle skills. It is also interesting to realize that apparently, there are two pretty distinct groups: one group of top players who have poor sliding tackle skills (a plausible hypothesis would be that this group mainly consists of attackers) and one group of top players that have good sliding tackle skills.

```
In [ ]: #Create histogram for attribute "finishing" with kernel density estimation (KDE)
        labels = ['Top 25%', 'Bottom 25%']
        sns.set_style("white")
        sns.distplot(top_players['finishing'], bins=20)
        sns.distplot(bottom_players['finishing'], bins=20)
        sns.despine(left=True)
        plt.legend(labels, loc = 'upper left')
        plt.xlabel('Finishing')
        plt.title('Distribution of rating for players on attribute "Finishing"', fontsize=14)
```

```
In [ ]: #Create histogram for attribute "sliding tackle" with kernel density estimation (KDE)
        sns.set_style("white")
        sns.distplot(top_players['sliding_tackle'], bins=20)
        sns.distplot(bottom_players['sliding_tackle'], bins=20)
        sns.despine(left=True)
        plt.legend(labels, loc = 'upper left')
        plt.xlabel('Sliding Tackle')
        plt.title('Distribution of rating for players on attribute "Sliding Tackle"', fontsize=1
```

```
In [ ]: #Print median for attribute "sliding tackle" in two groups
        print(top_players['sliding_tackle'].median())
        print(bottom_players['sliding_tackle'].median())
```

Plotting the distribution of age of the top players and the bottom player (to be precise: the distribution of age at each moment a player is rated as top 25% or bottom 25%, which means that the same player can appear in this distribution more than once) learns that top players are -on average- older than the bottom players. This can be explained by the fact that it will take some years before you reach your top ranking, while the younger bottom players still have room to grow (even with the filter on low potential).

```
In [ ]: #Create histogram for age with kernel density estimation (KDE)
        sns.set_style("white")
        sns.distplot(top_players['age'], bins=20)
```

```
sns.distplot(bottom_players['age'], bins=20)
sns.despine(left=True)
plt.legend(labels, loc = 'upper left')
plt.xlabel('Age')
plt.title('Distribution of rating for players on age', fontsize=14)
```

As said before, the same player can appear in the dataset multiple times. For the FIFA scores and age, this is not a problem, because they correspond to a certain moment in time (no player will be constant throughout his career). For height and weight, however, the data can be skewed if these constant values are included multiple times. Therefore I will filter the data for unique player names.

```
In [ ]: #Filter data on unique field players, leave ranking date February 2007 (see earlier comm
        filtered_all = players[(players['date'].dt.year > 2007)]
        unique_all = filtered_all.drop_duplicates(['player_name'])

        #Descriptive statistics for weight and height for field players
        unique_all[['height', 'weight_kg', 'bmi']].describe()
```

```
In [ ]: #Filter data on unique goalkeepers, leave ranking date February 2007 (see earlier commen
        filtered_g = gk[(gk['date'].dt.year > 2007)]
        unique_gk = filtered_g.drop_duplicates(['player_name'])

        #Descriptive statistics for weight and height for goalkeepers
        unique_gk[['height', 'weight_kg', 'bmi']].describe()
```

It is clear to see that goalkeepers are taller and heavier than field players, on average. This comes as no suprise: at least in the Netherlands, talent scouts look for tall goalkeepers. My favourite football club Ajax even does a hand measurement for young goalkeepers to predict if they will reach a minimum height set by the club.

```
In [ ]: #Create histogram for height with kernel density estimation (KDE), compare field players
        sns.set_style("white")
        sns.distplot(unique_all['height'], bins=15, label='Field players')
        sns.distplot(unique_gk['height'], bins=15, label='Goalkeepers')
        plt.legend(loc='upper left')
        plt.xlabel('Height in centimeters')
        plt.title('Distribution of height: field players vs goalkeepers', fontsize=14)
        sns.despine(left=True)
```

Let's see if there is also a difference in weight and height between top and bottom players. The tables below indeed show a small difference in weight and height for top players compared to bottom players: the top players are a little lighter and smaller than the bottom players, on average. Although I do not have the means at this point to see if this difference is significant, I suspect that (also taking the BMI into account) that there is no significant difference in weight and height for the top 25% players compared to the bottom 25% players.

```
In [ ]: #Decriptive statistics for unique players in the top 25% player group
        filtered = top_players[(top_players['date'].dt.year > 2007)]
```

```
        unique_top = top_players.drop_duplicates(['player_name'])

        unique_top[['height', 'weight_kg', 'bmi']].describe()
```

```
In [ ]: #Decriptive statistics for unique players in the bottom 25% player group
        filtered_b = bottom_players[(bottom_players['date'].dt.year > 2007)]
        unique_bottom = filtered_b.drop_duplicates(['player_name'])

        unique_bottom[['height', 'weight_kg', 'bmi']].describe()
```

There is one interesting (categorical) variable left to which the two player groups can be compared: preferred foot. Is there a difference in footedness between top and bottom players? This is a pretty difficult question to answer because to start with, there are no conclusive statistics on the percentage of footedness for all people but as a reference, I used the 19% leftfootedness found on Wikipedia. Comparing this statistic to the statistics below, leftfootedness does seem to be more common among football players compared to the general population. Leftfootedness is seen a little more often among bottom players in this dataset.

At this moment in the course, I am not entirely sure which statistical method I can use to check if this difference is significant or not but I suspect it is a Chi square test. If I run a Chi square test online, I do find a significant difference in footedness between the whole population and both top and bottom players. This could be an indication that professional football players are more often leftfooted than people who do not play football (or, at least not on a professional level).

```
In [ ]: print(unique_top['preferred_foot'].value_counts(normalize=True))
        print(unique_bottom['preferred_foot'].value_counts(normalize=True))
```

Conclusions

After trying many roads and angles within this extensive dataset, I singled out four research questions. Not every research question were answered to my full satisfaction. For example: I hoped to create a sharper image of the characteristics of top football players than I eventually found in the data. On the other hand, it was nice to see some research questions answered and visualized like I suspected beforehand, for example with the question about the relative birth effect.

A summary of my findings:

Matches * The much debated home advantage where football teams benefit from playing in their own stadium does seem to exist. There are differences between the various leagues, though. The cause of these differences is not clear. The hypothesis that the home advantage is larger in larger countries (with longer travel distances between clubs) is not backed up by the data. * Matching with my own observations between 2008 and 2016 ;-), the data did indeed show that Ajax was successful in terms of points earned and goals scored, compared to eternal rivals PSV and Feyenoord. However, Ajax did not entirely live up to its reputation of having an offensive playing style because the team did not score the most goals in the majority of the seasons in the dataset.

Players * Looking at the months in which the professional football players in this dataset are born, it does seem that the relative age effect is present, because an overrepresentation of players born in the first months of the year can be observed. However, at this moment in time I do not have the tools to test if this overrepresentation is statistically significant or not. * There is a strong correlation between the overall FIFA rating of the player and his reactions, which means that high overall ratings often go together with good reactions. Furthermore, the top 25% players have a

larger mean rating on long shots and shot power, followed by ball control, reactions, positioning, vision, volleys and curve (as compared to the bottom 25% players). An important thing to keep in mind though is that these characteristics are no hard numbers but subjective ratings created for a video game. An interesting article on how the ratings are calculated can be found here. * Goalkeepers are -on average- taller than field players. When comparing top to bottom players, there is also a small difference in weight and height: the top players are -on average- a little lighter and smaller than the bottom players. Although I do not have the means at this point to see if this difference is significant, I suspect that (also taking the BMI into account) that there is no significant difference in weight and height for the top 25% players compared to the bottom 25% players.