Visualization of Football Data

Rasmus Bo Adeltoft
Sebastian Seneca Haulund Hansen
Steffen Berg Klenow
Christian Bjørn Moeslund
Andreas Staurby Olesen
Henrik Sejer Pedersen

Supervisor: Marco Chiarandini

26th May 2016

1 Abstract

2 Preface

Contents

3 Introduction

Data is collected at a rapidly increasing rate in all fields and it becomes necessary to present data in different ways in order for humans to make sense of it. One way to do this is through data visualization. Visualization can help human's understanding of large data sets, as the data can be summarized very effectively, and patterns can quickly be recognized by humans. When making visualizations it is important to understand how the human cognitive system works, such that visualizations can be designed to make it easier for humans to understand the data. In order to do this, we will apply principles from the field of visualization to present football data. We will use tools such as R to process data and plot static visualizations, and use D3 to make interactive and dynamic visualizations.

Specifically, we will do this both by making visualizations that can help explore the questions that we present below, and by doing exploratory analysis such that new patterns can be discovered. The specific questions that we will be investigating are:

- How does a team evolve throughout a season in terms of goals, points, etc.?
- How does a team's playing style (for example passes, possession and tackles) change throughout a match?
- How does a winning team differ from a losing team?

During the visualization process we will consider different visualization techniques and choose a suitable one based on principles and analysis tools given by Tamara Munzner in "Visualization Analysis and Design" to make sure that the data is presented in an accurate and easily understandable manner. This includes considerations regarding the human cognitive system.

4 Theory

4.1 Design Process

This section describes the typical work flow of a data scientist. We will focus on the following four faces: Preparation, Analysis, Reflection and Dissemination.

The process of getting the data, understanding the data and produce results is an iterative process. The process is seen on Figure 1.

- The first face is the preparation face. Here the you have to acquire the data, that could be from hard disks, servers, through an API ect. Where to store and how to organize the data files should be considered, so it is easy to replace the right files if the data gets updated. Then the data should be cleaned, meaning removing tuples with missing values, changing the formatting, sorting the data ect.
- The second face is the analysis face. Here the data is analysed to get more information about it. This is an iterative process, where the you create and run scripts, look at the output, maybe find some mistakes, debug these and run it again.
- The third face is the reflection face. Here the output results is discussed, for example by making comparisons between outputs, and exploring alternatives.
- The fourth and last face is the dissemination face. Here the the results are reported and maybe published in a report. [Guo(2012), Chapter 2]

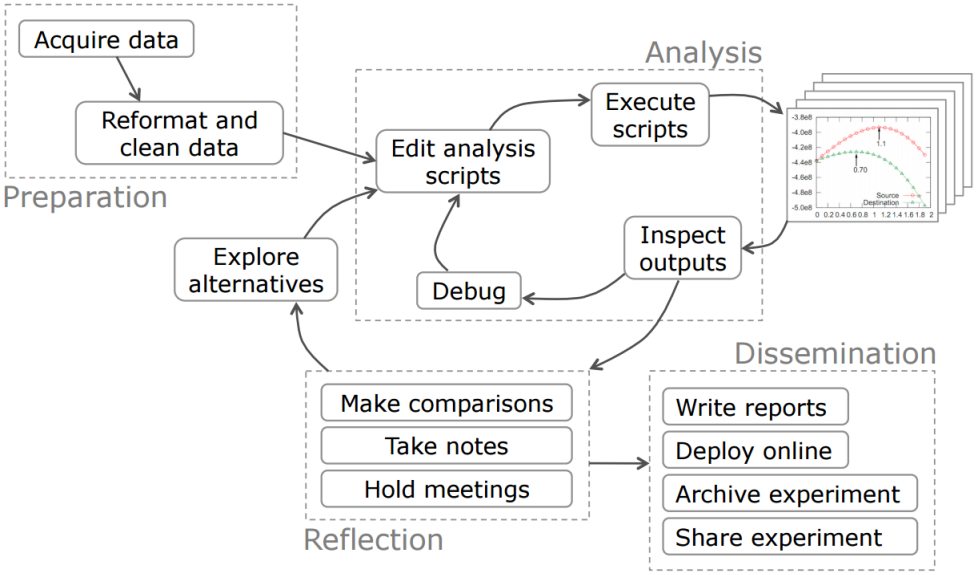

Figure 1: The model showing the iterative process[Guo(2012), Chapter 2]

4.2 Design

4.3 Data Types and Data Sets

4.4 Idioms

4.5 Analysis and Complexity

4.6 Facets and View Manipulation

When creating visualisations it is not always enough with one idiom to present the data in a understandable way. One way cram more information into the idiom, but still keeping the number of variables low is to either manipulate the view of the idiom or to facet into multiple idioms.

4.6.1 View Manipulation

By creating multiple views in an idiom, the idiom can contain more information, without clutter. The different views can include changes like switching between different idioms, changing the viewpoint, changing the order of the data, changing the number of items showed and so on. For example you can change the way the data is ordered by sorting the data by different variables. This is very powerful because of spatial position being the highest ranked visual channel. Many view manipulations is based on animation. Animating has a trade off, the cognitive load can be very high if too many elements change. This means that we get low cognitive load if either some elements are static and others moving, or some groups of elements are static and others moving. If few elements change by a graduate transition, the viewer can keep the context between the two views. [Munzner and Maguire(2015), Chapter 11]

Selecting one or more elements is common in many interactive visualisations. The result of selecting some elements is then some change in the view. It has to be considered which elements the user can select, and how many the user can select. Choosing how to select items is also subject which has to be considered, clicking to select, hovering over

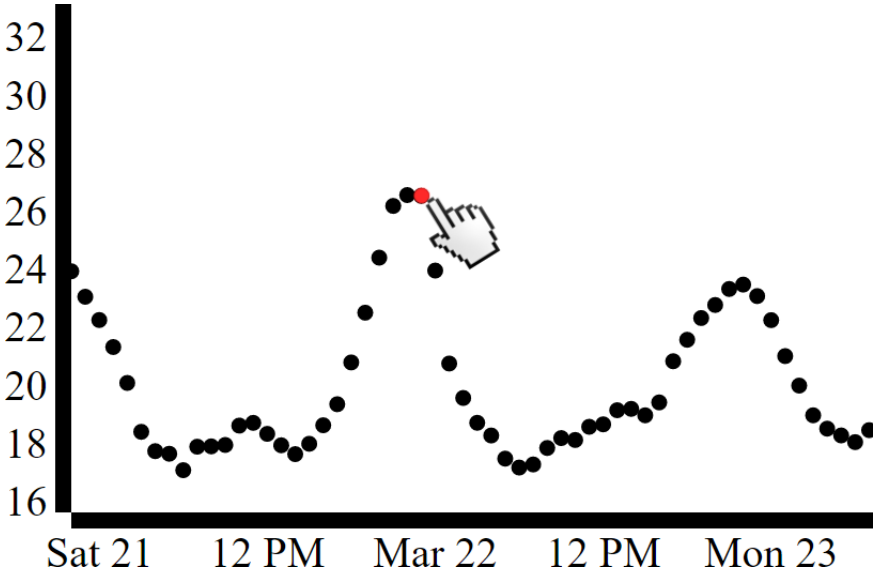

Figure 2: Highlighting the selected element

some element with the cursor or something else. Changing the view by highlighting some element and creating pop out could be done by changing the channel, for example the color, the size, the outline or the shape of the element. This change should of course be so dramatic that the element clearly stands out from the rest. Look at Figure 2 for an example of highlighting. [Munzner and Maguire(2015), Chapter 11]

Another option for an interactive idiom is the ability to navigate the view by changing the viewpoint. Here we think as if we have a camera pointed at the view. We can then change the view by zooming, panning or rotating the the camera around its own axis. There are two kinds of zooming, geometric zooming and semantic zooming. Geometric zooming is straight forward making some elements come closer to the camera. With semantic zooming the not only the size of the elements change, but the semantics too. Semantic zooming changes what is shown, and maybe the representation of it. For example zooming semantically could view more detail about some element showing new information about it. Navigation could also be changed through reduction of attributes, by slicing, cutting or projecting. These are all dimension reduction techniques. To slice, a specific value at a dimension is chosen, and only elements matching this value is shown. A cut is made by placing a plane in front of the camera, all elements in front of the plane is not shown, in this way it is possible to explorer elements behind other elements, or looking inside 3D objects. Projection is done by eliminating some dimension but still showing all the data. This is similar to what the human do when looking at a 3D object.

4.6.2 Facet

Faceting is splitting the view up into multiple views or into multiple layers. One of the main reasons to facet is to compare views. This is much easier than comparing two views in a changing view, because we do not have to remember the prior view, but continuously can compare them. Another reason to facet is to gain more information about the data through a multiform design, where data is shown using different encodings. By having multiple views more attributes can also be shown. Of course when you have multiple views shown beside each other, each view has less space, which is one of the trade off's and why having multiple layers on top of each other, and switching between them sometimes is

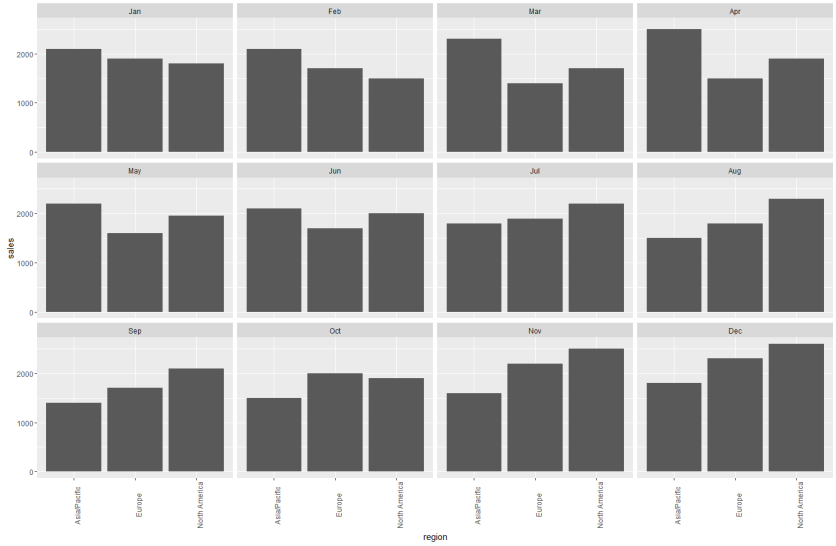

Figure 3: An example of small multiples

better. If having juxtaposed views it might be interesting to link the views. This could be done sharing the data, sharing the visual encoding, synchronizing the navigation or highlighting.

Each view could show different subsets of the data, or having different viewpoints like the classic overview in one view combined with detail in another view. Having multiple views sharing encoding but showing different parts of the data is called small multiples and is often structured in a matrix. This could be an alternative to animations, where we lay out all the frames. The cognitive load is smaller with small multiples, and it is easy to go one frame back or forth. An example of small multiples is seen at Figure 3. Instead of juxtaposing the views, it is a option to stack them into a single frame. The views should have the same horizontal and vertical extend and blend blend together as one frame, by being transparent where there are no marks. The problem with stacking is distinguishing between the layers. This is easy with only a few layers, especially if the layers use different visual channels. But distinguishing between more than three layers, can be a real challenge. [Munzner and Maguire(2015), Chapter 12]

4.7 Exploratory Data Analysis

Exploratory data analysis, EDA, is a philosophy about how one can approach the analysis of a data set. It is not a fixed collection of tools that one can use to analyze a data set, but it is a general approach which promotes looking at data in different ways without having any assumptions. It is usually used when one receives a new data set, and wishes to learn something about the underlying structure of the data set, or wishes to discover outliers or trends in the data. In general, EDA is about looking at the data that is presented to one, in many different ways. Both visual and non-visual methods are used in EDA. Examples of non-visual methods are simply to calculate the mean, median, variance, quartiles etc. of the data, while visual methods could be a boxplot, scatterplot, which quickly allows one to discover outliers and trends, or even a simple bar chart. The boxplot is a quick way to present some of the calculated properties mentioned above, such as the median and some of the quartiles. The boxplot was actually developed by John Tukey, who is widely regarded as one of the big promoters of EDA, and it was presented in his book “Exploratory Data

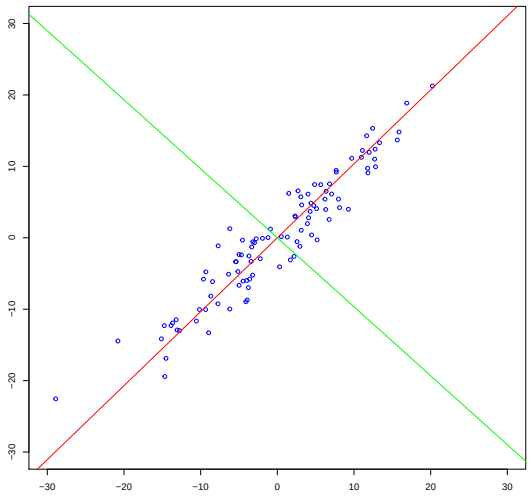

Figure 4: Some data set plotted, red line is PC1, green is PC2

Analysis", as a method one could use while doing EDA. Another approach one can use within EDA is principal component analysis, PCA, which is described in another section.

When doing EDA, one will often be able to use the acquired knowledge to formulate new hypotheses about the data, which can then be used to make more specific visualizations or calculate more specific properties of the data set, but it is not a guaranteed outcome. EDA will not always produce new knowledge, which is not a flaw in the approach, but rather a natural consequence of the approach.

4.7.1 Principle Components Analysis

Principal Components Analysis, or in short PCA, is a way to take high dimensional data and reduce it without loosing much information. It is difficult to find clusters, similarities and differences in high dimensional data, but by doing PCA it becomes much easier because of the dimensions reduction. [Smith(2002)]

If you have you multidimensional data plotted in a coordinate system, PCA finds the vectors which describes the most variance in the data. These are called the eigenvectors. The eigenvector describing the most variance is the first principle component (PC in short), the eigenvector describing the second most variance is the second principle component and so on. The PC's are perpendicular. Figure 4 shows a scatter plot with first and second PC drawn as the red and green line.

To find out which variables have the most influence on a PC, we can look at the coefficients of the PC, meaning the coefficients of the eigenvector, the higher the coefficient, the higher the influence. When having found the principle components, it is time to find out which to keep. This can be done by looking at a scree plot. This is a bar chart showing the variance for each of the PC's. Typically it is interesting to look at the first two PC's. We then look at the data expressed in terms of the principle components we have chosen. If we plot the derived data as a scatter plot, with for example the first PC on the x-axis and the second PC on the y-axis, we can easily point out the clusters of the data if there are any. This way we can look at the data in fewer dimensions but without much loss of information. In section 5.1 we do principle components analysis on some of our own data.

4.7.2 Linear Regression

4.8 Tools and Technologies

4.8.1 R

R is an open-source programming language and is often the go-to in fields such as statistics, visualization and other branches concerned with huge amounts of data. The language is heavily inspired by *S* developed at Bell Laboratories and thus one may find them to be quite similar. Being an open-source project under *GNU*, *R* is free to use both personally and commercially.

Even though *R* is often used as a tool to explore data, *R* is in fact a true programming language allowing the user to write and reuse procedures. Doing so, eases the work flow working on datasets with a reappearing structures in the sense that once a procedure is defined, it can easily be reused. Due to the fact that it combines the best of both world in statistics and programming it is a good candidate for areas such as machine learning too. Other than being a strong environment in itself, offering many standard operation, one of the true strength in *R* is its extensive range of packages. Being open-source, developers are constantly adding new packages to the environment extending the functionality of *R* whereof most of them are also free to use. The packages are what makes *R* really shine, as it adds some crucial functionality which has undoubtedly contributed to its mass popularity. Some of the packages such as *knitr* allows the user to write full reports alongside their results an convert them to various formats. Other packages makes some procedures less complicated by wrapping them into single methods, making *R* feel slightly more like a high-level programming language than it already is and allows the user to get results faster. Another strength of *R* which aided it in gaining mass popularity amongst the scientific community is its outstanding possibilities to visually display data. Other than having an array of different options in regards of how to visualize the data, being a programming language, one can alter how the specific visualization will look, making it highly customizable and almost limitless to the advanced user. Once again, packages extend the integrated functionalities and some mentionable ones would be *ggplot2* which makes it easier to plot data and takes care of the more complicated aspects such as drawing legends or making multi-layers. Another one would be *Shiny*, which also allows the user to make dynamic visualizations and add components to alter the plotted view.

Before visualizing data one has to clean and reformat it in respect to the preparation phase described by Jonas Schöley. *R* offers a range of different options when reformatting and supports a various number of data types to store an alter the data within. Other than supporting some basic data types such as arrays and lists, it also supports *vectors*. A vector is simply put a sequence of elements in an array-like structure. Vectors can be accessed through indexes like an array but the size of it will grow an shrink according to how much data it contains. Vectors have multiple purposes and may be used to describe a sequence of values along an axis in a coordinate system or may just be used as a middle-ground to construct some other rather sophisticated data structures such as the *data frame*. Data frames are commonly used as the go-to input type in different methods and thus, the user will often need the data converted into this format at some point. Data frames are very much like a matrix structure-wise but they are slightly more general purpose in the sense that each column does not need to have the same data type. Having this attribute makes it a very good candidate for storing large dataset, as it is more flexible in terms of what each cell actually contains. Often R will have an built-in method to import a data set into

the environment and convert it into a data frame but sometimes a package will need to do the trick. However, as some datasets may have a more complicated structure than a data frame, for instance by having multiple in-going layers, a direct conversion is not always an option and some manual work must be done to built the dataset. When a data frame is constructed, the user often want to restructure is. Unfortunately, *R* does not offer an easy and intuitive way itself to get such job done. Luckily however the packages *tidyr* and *dplyr* comes in handy when facing such issues. *Tidyr* helps obtaining tidy data set, meaning that each column represents a one variable only and each row contains a single record. To obtain a tidy dataset, the package allows operations on a data frame to reshape the structure which may be to split column into two or converting a row into columns. *Dplyr* on the other hand allows the user to do operations on the data frame like selecting single columns, add new ones and much more. Combined, these to makes up for an excellent framework to sanitize data before having to visualize it.

However, even though R in some cases are indeed an excellent choice it also have its limitation where other technologies may be better of. The way *R* is built makes it very prone to heavy memory costs partly because it allows the user to do single line execution and thus have to save all assignments of variables. Singly line execution is indeed a neat feature whilst developing code but may not always be desirable. In terms of speed it is very often criticized for being really slow which is often a rather common trait in high-level programming. In a test done by Jacob Simmering he found that *R* performed 270 times slower than *C* whilst looking for prime numbers. Some factors such as the fact that `C` is compiled at run-time may have an impact on the result for sure. However, another language which is neither compiled is python which turned out to have 17 times faster performance relative to *R*. Some tricks can however be made to boost the performance of *R*, as Simmering purposes a *byte code compiler* which can increase the speed of *R* tremendously. In regards to competitors, some users tends towards technologies with a more WYSIWYG focused approach such as *Excel*. Whilst R is really good at exploring data once the method to get there are defined, it takes a while to wind up the configuration. Spreadsheets like *Excel* are rather fast at getting smaller tasks done and are arguably visually prettier for presentation purposes which makes it a viable option in some scenarios. However, excel may fall short if the tasks become more complex, for instance if the data has to be formatted before being visualized or has to be presented in some highly customizable way where the bounds of *R* are almost limitless. *Python* is another viable option when doing statistics and is a more programming focused solution. Both solutions are indeed viable options and it is a matter of taste to which one may prefer. As argued earlier, Python may win when it comes to speed and due to the fact that it is more programmer oriented language, small task which no one has done before may be accomplished more easily where *R* tends to be slightly more reliant on packages which is a doubled-edged sword.

4.8.2 D3.js

5 Results

5.1 Principle Component Analysis

In this section we do Principle Component Analysis on aggregated on one season of football data. The data includes observations like the number of goals, the number of shots, the number of passings and so on. The data is collected for each player, but we have summed it to each team. We would like to find any clusters by using PCA in R.

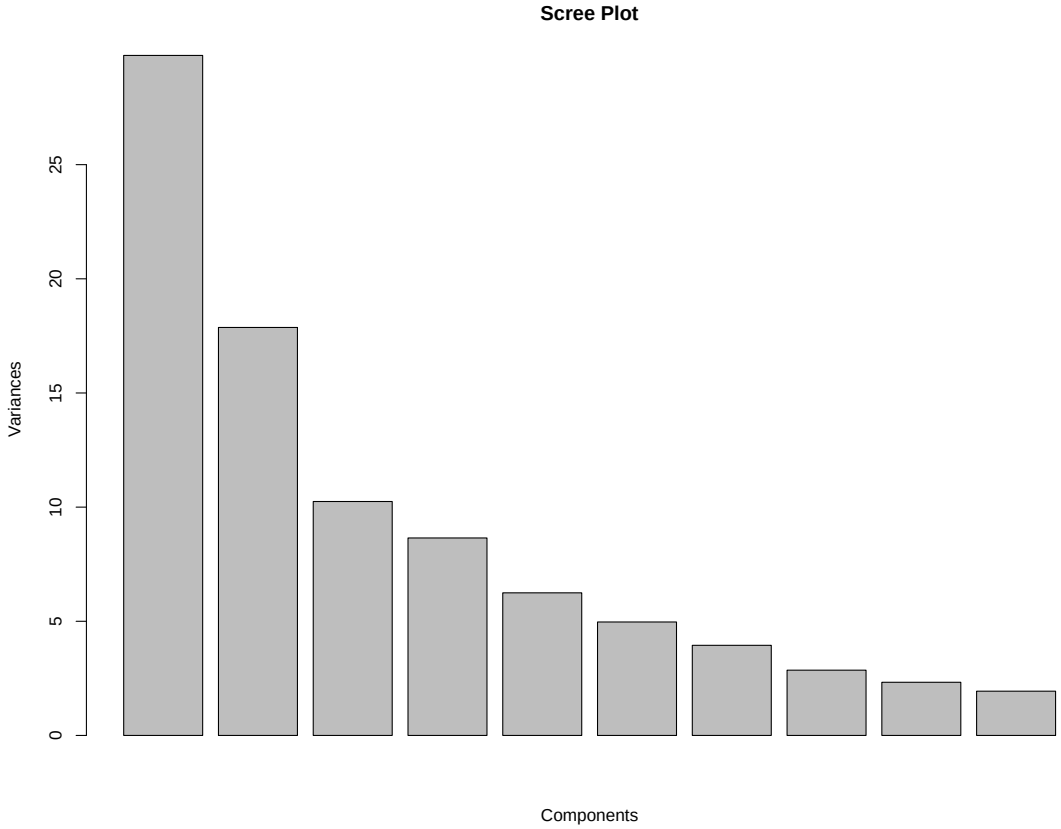

Figure 5: The resulting scree plot

The resulting scree plot from doing the PCA is seen in Figure 5. The scree plot tells us which eigenvectors have the highest eigenvalue, meaning which principle components describe the most of the data. In our case we find that the first two components describe a lot of the variance in the data, to some extend also component 3.

We know look at the loadings each of the components that we choose to keep. The loadings describe the importance of the different variables in relation to the chosen component. In Figure 6 we see the loadings for PC1. Being that PC1 is the component describing most of the variance in the data, the most important variables of PC1 is also the most important variables for the entire data set. The five most important variables in regards to PC1 is the "Sum.of.Chances", "Sum.of.2nd.Assist.To.Shot", "Sum.of.Fouls.Received.In.Second.Third.Part", "Sum.of.Assist.To.Goal" and "Sum.of.Shots.On.Target".

We can do the same with PC2 and PC3. These are seen at Figure 7. From the loadings in regard to PC2 we conclude that the five most important variables are the "Sum.of.50.50.Air.Challenges", "Sum.of.50.50.Air.Challenges.Won", "Sum.of.Crosses.On.Restart.Of.Play", "Sum.of.Successful.Crosses" and "Sum.of.Crosses". For PC3 the five most important variables are the "Sum.of.Fouls.Received.In.First.Third.Part", "Sum.of.Goals.With.Right.Foot", "Sum.of.Tackles.Made.Won", "Sum.of.Goals.With.Foot" and "Sum.of.Crosses.In.Game".

We can now plot the data points in regards to the principle components to find clusters. The data is sorted in regards to current position of the teams in the table. This means that we easily can find out if the top/bottom teams cluster. In Figure 8 the data is plotted in regard to PC1 and PC2. We do not have any clear clusterings, some teams are clustered in the middle but there is no correlation between the standings and the teams clustering.

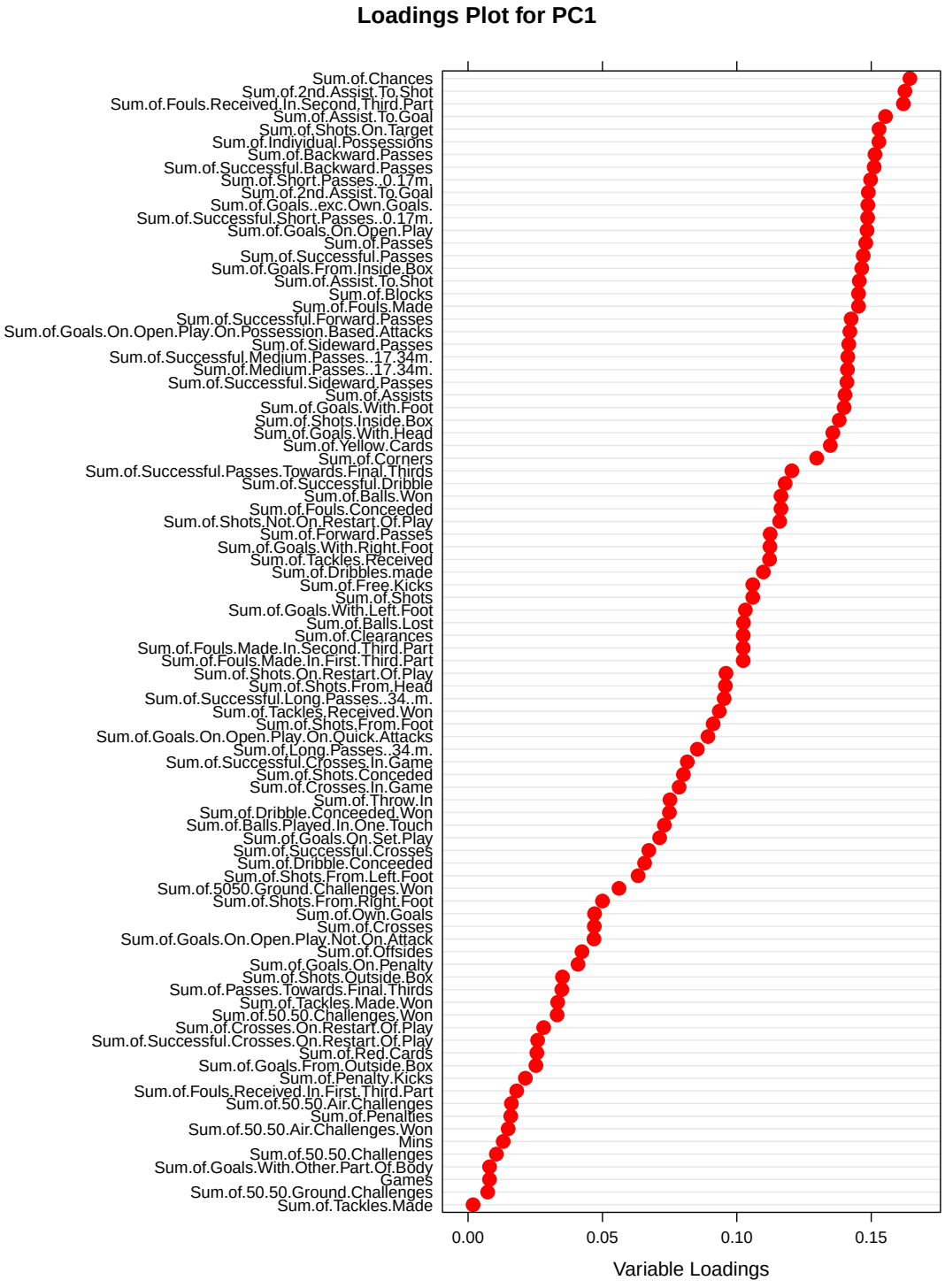

Figure 6: Loadings for PC1

Figure 7: Loadings for PC2 and PC3

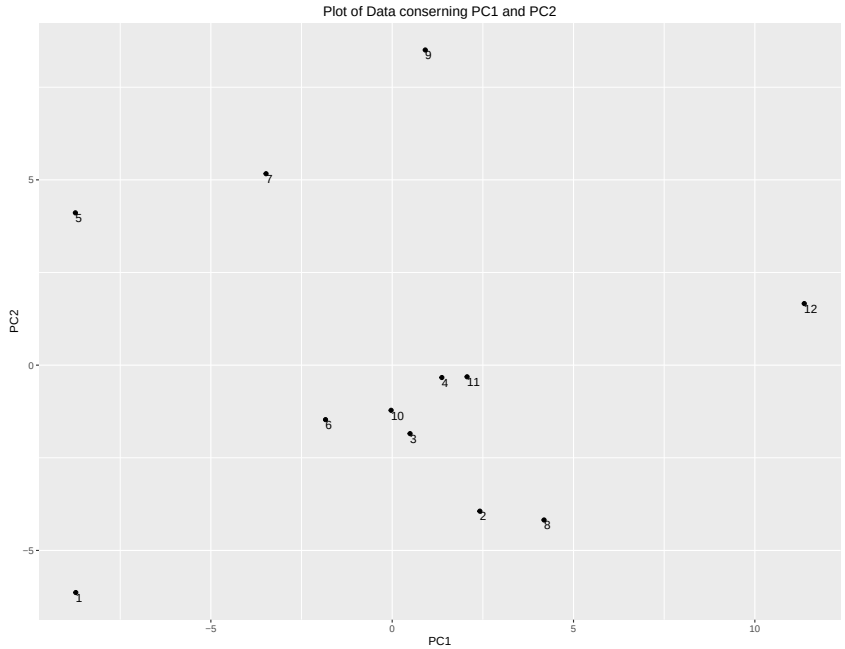

Figure 8: The data plotted in regard to PC1 and PC2

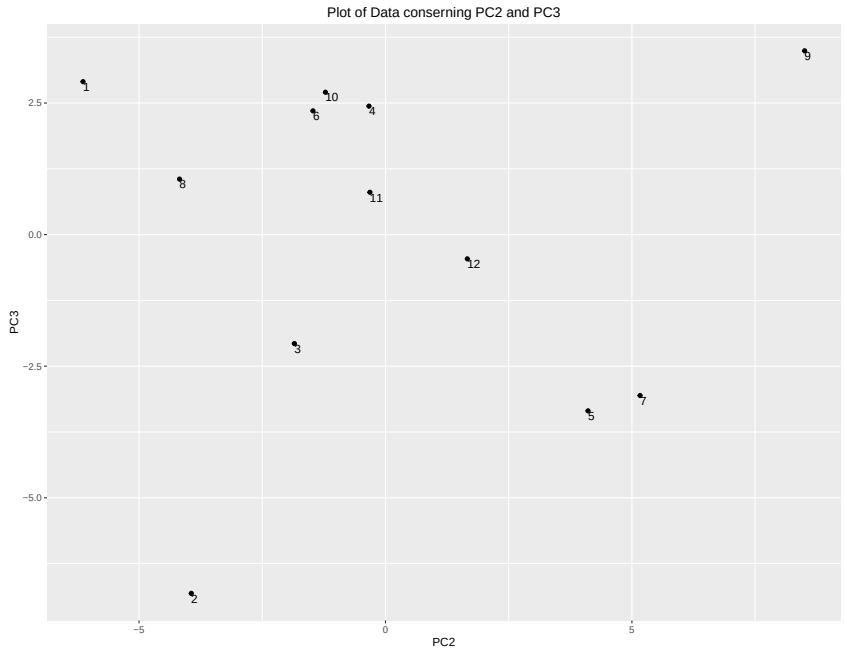

Figure 9: The data plotted in regard to PC2 and PC3

In Figure 9 the data is plotted in regard to PC2 and PC3. Once again we have a little cluster, but there is no relation to the standings.

The PCA has not giving any clusters, but it has giving us some of the most important variables of the dataset.

5.2 Success Rates

5.2.1 What-why-how

The visualisation shows the comparison of two teams' success rates on several points of measure in a radar chart combined with a bar chart. This is shown in Figure 10. Concerning magnitude channels we position the marks on a commons scale and also use the area of the radar chart for comparison. For categorising we use color hue.

The action of the idiom is to compare two teams to find differences/similarities, maybe in relation to the placement in the league table, and to analyse where the teams differ. The target is to find extremes and outliers.

This is done by faceting into two views. In the left view we have a radar chart and in the right view we have a bar chart. The user can select two teams to compare in the radar chart. The teams' success rates in relation to shots, tackles, air challenges, passes and dribbles, is shown on the radar chart. When pressing one of the team's values in one of the measurement points the bar chart comparing all teams in this measurement is shown. The views are linked together by synchronizing the highlighting. The team chosen in the radar chart gets highlighted in the bar chart. The highlight in the bar chart creates pop out by having this bar coloured and all others black. The raw data is reduced by filtering and by aggregating. Only some variables are chosen and the values for each team is calculated from the individual players' values. The radar chart is manipulated first off by selecting two teams to compare. If the user hovers over a value, all the team's values together with the area of the chart is highlighted, by changing the opacity of the other team.

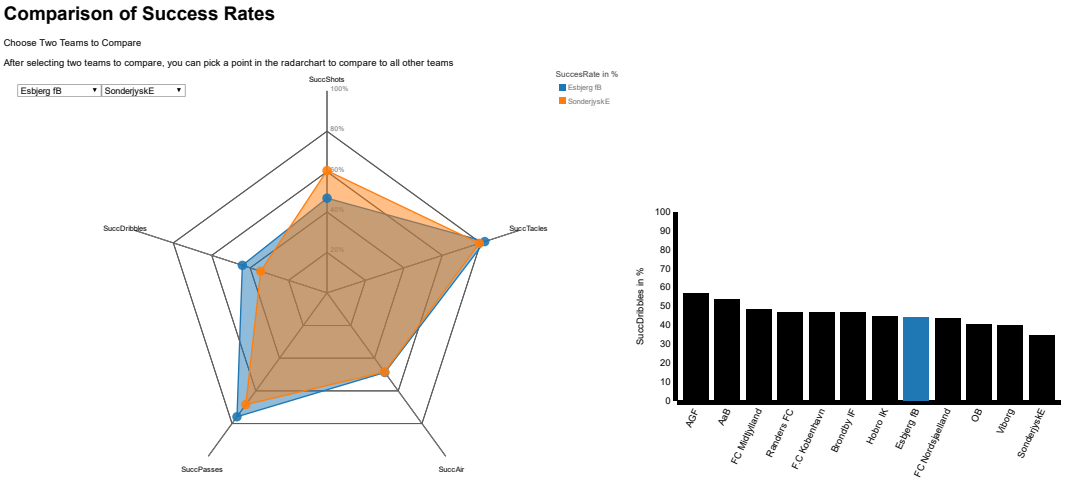

Figure 10: Idiom showing the comparison of the success rates using a radar chart for comparing two teams and bar chart for comparing with all other teams in the league

5.2.2 Code

The R code is quite simple. In broad strokes it does the following: Read the data file, convert columns to be numeric, sum the data by team (before player wise), add and calculate the success columns, remove the unnecessary columns, restructure the data to a format that fits the radar chart and write three files, one containing the names of the teams,one containing the data in one format and one containing it in another format. The data is exported in two files to suit both the radar chart and the bar chart.

In JavaScript we create the selectors, where the user chooses the teams. If two teams are chosen the radar chart is drawn. The radar chart is based [d3noob(2013)] and modified to our needs. One of the modifications is that we draw a bar chart if the user presses a value, represented by a circle. The function which draws the chart takes the data variable to visualise as an argument together with the selected team. The data is then loaded and drawn making sure that the data is sorted for better comparison, and that the right team is highlighted.

5.3 Nationality of players

5.3.1 What-why-how

This visualisation shows the nationality of the players on the different teams in the league. It also shows the distribution of the foreign players among the other countries. This is done by a sankey chart, where the teams are sources, the targets are either Denmark and Foreign. On the second level the source is Foreign and the target is the other countries. The idiom is shown in Figure 11.

The actions of this idiom is to summarise the players nationality, to explore and compare the different teams to find similarities/differences between the teams in comparison to the standing in the table.

This is done by an interactive sankey chart. The view is manipulated by highlighting when hovering over a link, where we also get the percentage of players in this connection in

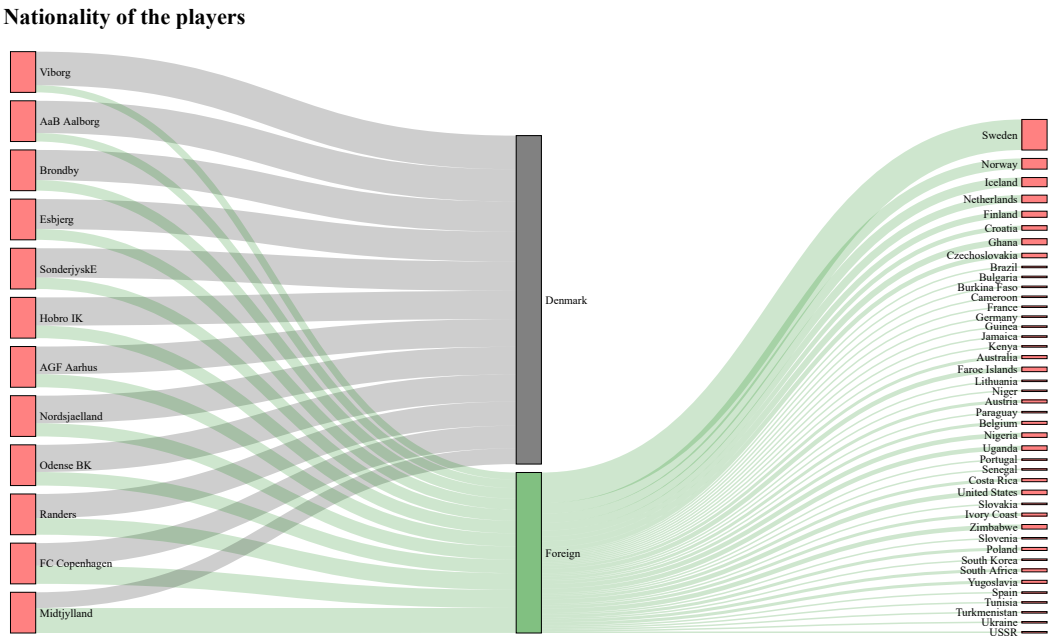

Figure 11: Sankey-Chart showing the distribution of danish vs. foreign players on the different teams

form of a text box. The raw data is reduced by filtering by only focusing on the nationality of the players. The players are categorised into the different bins by using hue and region.

5.3.2 Code

In R we load in the data file, which now is a .json file. We then find the data we want to look at in the .json file, and convert it into a data frame. We then sum the nationalities of each club. We then create a copy of the frame. In the original frame we set all the sources to be foreign where the target is not Denmark. In the second frame we set the targets to be Foreign where the target is not Denmark, and then remove all rows where the target is Denmark. In both frames we sum the values for each source-target pair. We then combine the two frames and exchange the value being the number of players, to now being the percentage of players. Finally we write the data to a .csv file.

The sankey chart itself is based on [Bremer(2016)] and modified to our use. We modify the placement of the nodes to have multiple levels, and clearer color scheme with clear differences between the links.

6 Discussion

7 Conclusion

8 Usability

9 Bibliography

References

[Bremer(2016)] Nadia Bremer. D3.js - radar chart or spider chart - adjusted from radar-chart-d3, 2016. URL `https://gist.github.com/d3noob/5028304`.

[d3noob(2013)] d3noob. Sankey diagram with horizontal and vertical node movement, 2013. URL `http://bl.ocks.org/nbremer/6506614`.

[Guo(2012)] Philip Jia Guo. *Data Science Workflow: Overview and Challenges*. PhD thesis, Stanford University, 2012. URL `http://purl.stanford.edu/mb510fs4943`.

[Munzner and Maguire(2015)] Tamara Munzner and Eamonn Maguire. *Visualization analysis and design*. CRC Press, Boca Raton, FL, 2015. ISBN 9781466508910;1466508914;.

[Smith(2002)] Lindsay Smith. A tutorial on principal components analysis, 2002. URL `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`.

10 Appendix

11 Process Evaluation