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

27th 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]

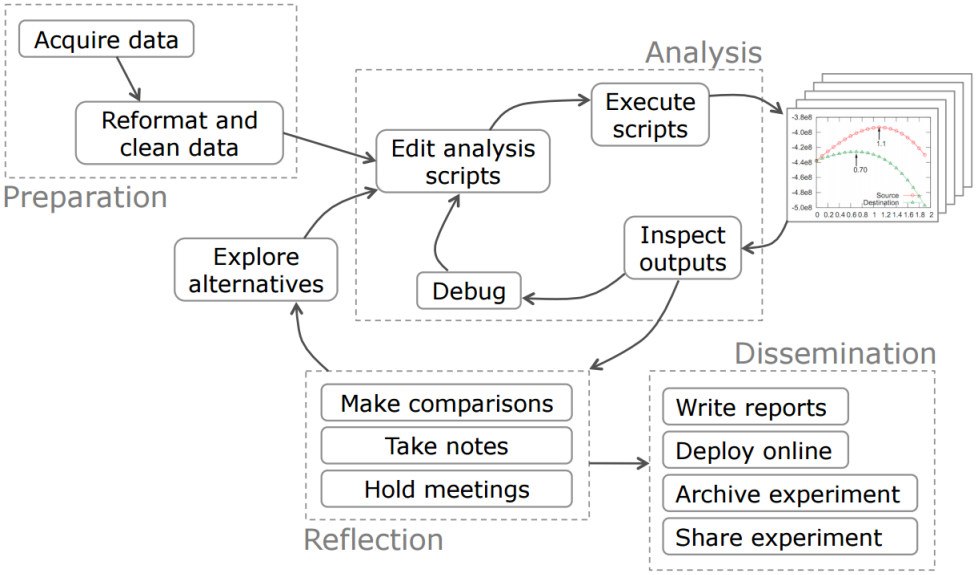


Figure 1: The model showing the iterative process[Guo(2012), Chapter 2]

4.2 Design

The following section explains the process of designing visualizations. The process follows the What-Why-How structure. Following is a brief explanation of the considerations a designer makes during this process.

The first question a designer has to answer is: "What?". What are the semantics of the data, what type of data is provided, what dataset is the data presented in? All of this is important, as understanding the specifics of the provided data is necessary for designing the most efficient visualization for this data. The efficient visualization designs for geometry are drastically different from those of tables. An example of geometry is the location on a football field where a given action has taken place, like a successful tackle. Here, a smart visualization could be illustrating where on the football field the most successful tackles are made, using a football field as a coordinate system. Here, the data type would be positions and items, describing what happened, and where. In a table, similar information could be stored, however, instead of positions, it would contain attributes. For instance, an item could be a specific player, and an attribute could be how many successful tackles he has during a game. A smart illustration here could be a bar chart, illustrating this player's efficiency (amount of successful tackles in games) compared to other players at his position, to help figure out who is the best player for certain situations where defending is key. After the data has been identified by the designer, the next question that needs answer is: "Why?". Why does this data need to be visualized? What is the target of this visualization? Is the goal of the visualization to consume existing data or to produce new information? This faze can be split in two: Actions and Targets. Actions are mainly comprised of analysing, searching and query. Analysing depend on whether the goal is to consume or produce data. Search describes finding the data that answers the questions that are being asked. Searching is dependent on whether the location of the data is known or unknown, and whether the target is known or unknown. There are four kinds of searching; lookup, where location and target is known. Browse, where location is known and target is unknown. Locate, where the target is known but location is unknown. And explore, where neither target nor location is known. After the search, the goal is to query the targets

at one of three scopes: identify, compare or summarize. Identify refers to a single target, compare refers to multiple targets and summarize refers to the full set of possible targets.

The other part of the "Why?" faze is Targets. Here, the designer identifies what the goal of the visualization is. To locate trends or outliers in datasets, identify correlations, the shape of objects, etc.

The final question is "How?". How is this visualization going to be designed, what design choices should be made to accomplish the goal of the visualization? Here, the designer should also decide the specifics of the visualization. In short, this is where the designer maps out how to arrive at the final visualization. How to encode the data, how to map it in the visualization, how to manipulate the visualizations for better visual efficiency, how to facet the data and how to reduce the data.

4.3 Data Types and Data Sets

In the book "Visualization Analysis and Design", Tamara Munzner describes why data semantics and data types are important. In short, these are vital for humans' ability to understand the data. The semantics of the data is its real world meaning. This is self-explanatory, by being what the data represent in the real world. A table with a row containing 5 different numeric values are useless unless the person(s) working on it knows what these numbers represent. The type of the data is equally important, as this is vital in understanding how to work with the given data. If it is a number, is it a quantity or an ID? This information is key, as adding two numeric values together makes sense for quantities, but would be useless for ID or code. There are five basic data types described in "Visualization Analysis and Design": Items, attributes, links, positions and grids. An item is an individual entity, like a row in a table, since a single row in a table only describes one unique item in the given table. An attribute is a measurable property, such as the base salary in different professions, a person's height and so on. A link is the relationship between items in networks, an example being the relationships between people in a city, such that a link describes the relationship between the citizens (Neighbours, family, acquaintances, friends...). A position is spatial data, describing a location in two-dimensional or three-dimensional space. Here, a great example is the earth, which is divided in latitude and longitude coordinates, such that every point on earth has a positional data, or coordinate. Finally, a grid describes the strategy for sampling continuous data in geometric and topological relationship between its cells. The four basic dataset types are tables, networks, fields and geometry. Other possibilities include clusters, sets and lists. However, it is common for these basic types to be combined in real-world situations. Figure 2.3 illustrates what these dataset types are consisting of. Following will be a short description of each dataset type. In a table, each row represents an item of data, and each column is an attribute of the dataset. Each individual cell contains a value for the pair of the item and the attribute.

Networks and Trees specify the relationship between two or more items. An item in a network is commonly referred to as a node, and a link is the relation between two nodes. A network with a hierarchical structure is called a Tree. Unlike general networks, trees don't have cycles, as every child has only one node pointing to it, referred to as its parent node.

The field dataset type consists of cells. Each cell herein contains measurements from a continuous domain. In a way, there is an infinite amount of values to measure. Examples of this is in the real world would be measuring the temperature. Finally, the geometry dataset type describes information about the shape of items. The items can range from one-dimensional lines to three-dimensional reconstructions of buildings and landmarks.

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 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.

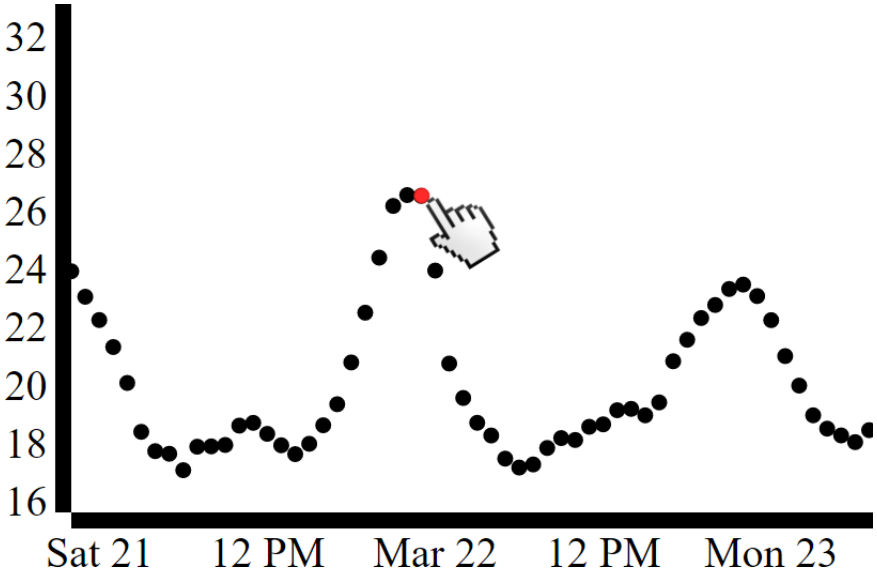


Figure 2: Highlighting the selected element

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 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

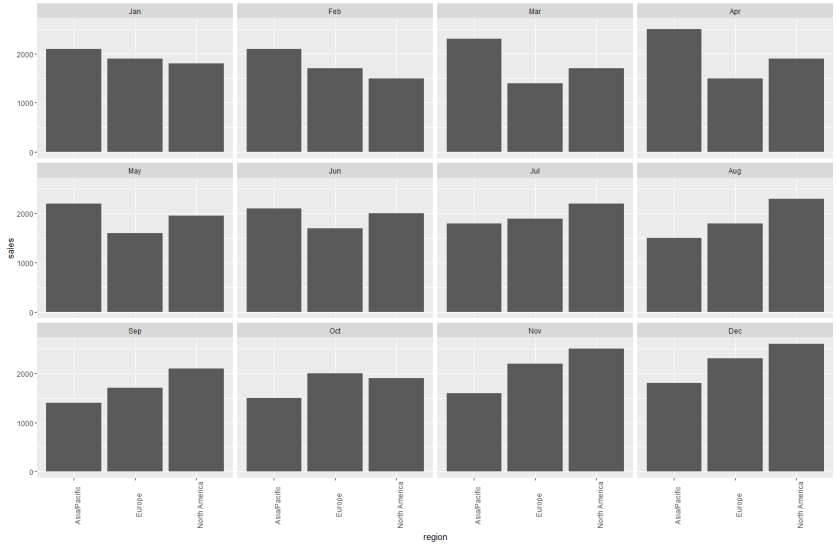


Figure 3: An example of small multiples

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 the quartiles. The boxplot was developed by John Tukey[Weimer(2016)], who is widely regarded as one of the big promoters of EDA, and it was presented in his book "Exploratory Data Analysis", as a method one could use while doing EDA.

In specifics a boxplot[Seltman(2016)] presents the following to the viewer, see figure 4.

Where a quarter of the data lies below Q1, a quarter is above Q3, and a half is between Q1 and Q3, where the median is centered between the Q1 and Q3. The area between Q1 and Q3 is often referred to as the interquartile range, IQR, which is calculated as Q3 minus Q1. The strength in this visualization is that one can quickly see the spread of the data. The bigger the IQR the more the data is spread, and the smaller the IQR, the less the data is spread. The boxplot also shows us the extremes and minimums of the data set, and the outliers. The lower whisker encodes the minimum value that is within 1.5 times the IQR subtracted from the Q1. Any values that is smaller than this will be represented as an outlier, which is a mark that is placed below the whisker. The same goes for the upper whisker. Boxplots are especially useful when you have multiple data sets that you want to compare, as you can place multiple boxplots next to eachother, and very quickly be able to recognize the differences between the data sets, as we have now encoded the data into something visual which is easier for the human cognitive system to process subconsciously.

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

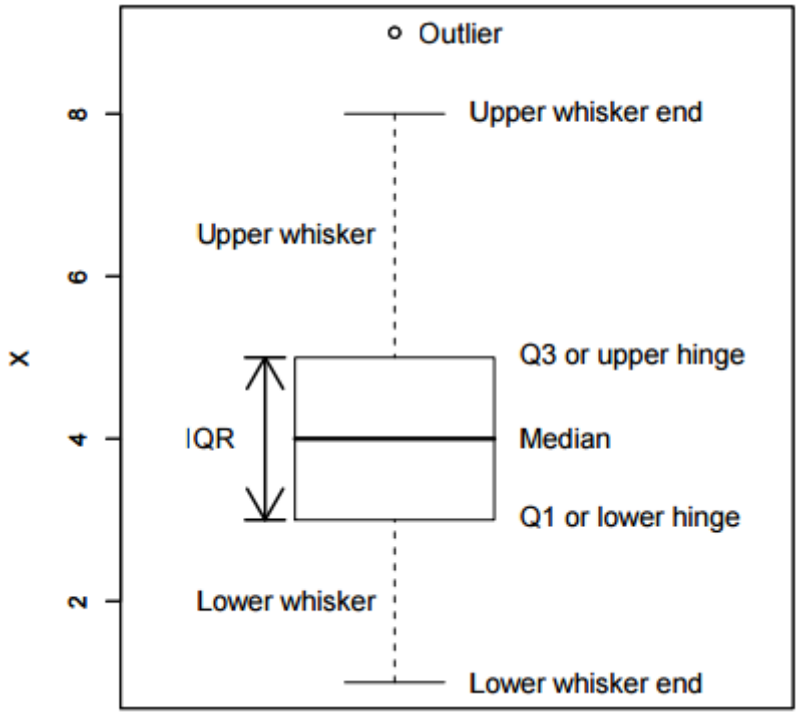


Figure 4: A boxplot[Seltman(2016)]

natural consequence of the approach.

When doing EDA, one is not limited to the "known" idioms/methods to visualize or look at the data. One can easily produce new kinds of visualizations which can be used to learn about the data.

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 5 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 6.1 we do principle components analysis on some of our own data.

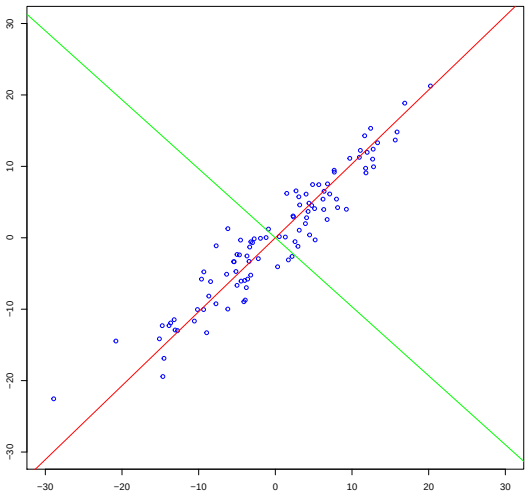


Figure 5: Some data set plotted, red line is PC1, green is PC2

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 Data and API

The data that we use in our visualizations come from two different places. The first place is from an excel sheet with data about the football teams in the "Super liga" (now referred to as the liga), which we were sent by Prozone (the company that is giving us access to their football data). That data is already somewhat processed and it is very convenient to work with it in R, as one don't have to do a lot of API calls. The second place that we get our data from is from an API that we have received by Prozone. The API works in this way: you go to their website and browse the different things you can request. When you have found what you need, you get a URL where you can put arguments in, for example specify a team id or the encoding format (XML or JSON). When that is done, you need to generate a signature, which is the current unix time plus a shared key and an API key which we have been given by Prozone. That string is then hashed using SHA-256, and appended onto the request URL along with the shared key unhashed. Then you simply send a HTTP request, and you receive the data. The request link is only valid for a short period of time, meaning that if anything interesting has to be done, the requests have to be generated automatically. For this purpose, a function has been implemented in R which automatically generates the request URL and returns a data frame, when given the first part of the URL which describes what data is wanted. The function is especially useful when one has to retrieve large amounts of data from the API, as some of the data is very tedious to get from the API.

The data that we have access to through the API is things such as a list of all football teams in the liga, their rankings and so on and data from specific matches. An example of what one has to do to get all the goal attempts throughout the season is: First get a list of all football teams in the liga. Then for each football team get a list of the matches that they have participated in. Then for each match retrieve the events in that match, and then put all of those events into one big data frame, and then filter the events such that only goal attempts are left. What is important to note about the API is that one cannot simply say for example "give me all matches where there is atleast 5 goal attempts within 10 minutes". One has to programmatically make multiple calls to the API and based on that data, decide what other calls should be made.

A short list of some of the data we can get through the API:

- Football teams in the liga
- Statistics about a football team (goal attempts, points, number of players, city of origin etc.)
- Players on a team

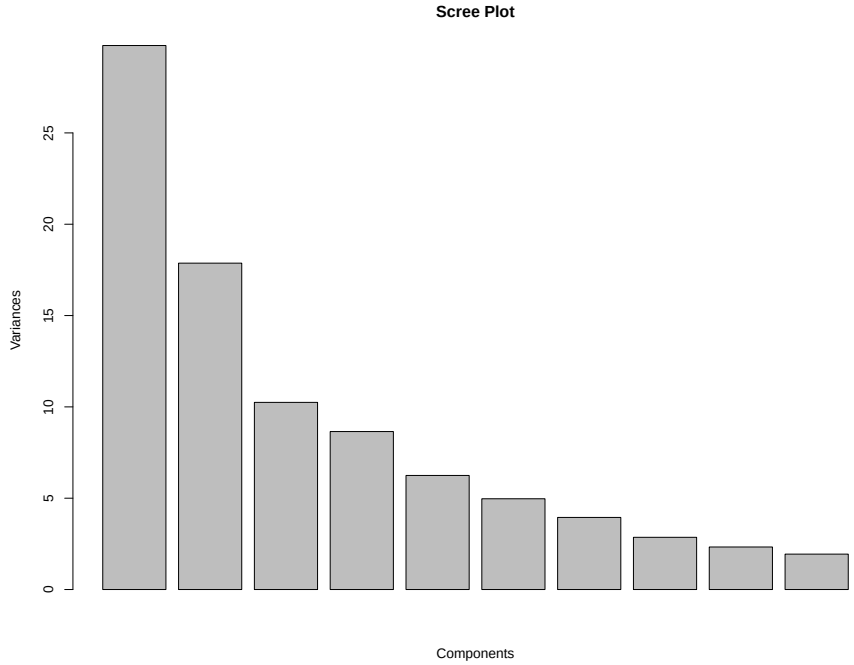


Figure 6: The resulting scree plot

- Information about a player (height, weight, age etc.)
- Matches a team has participated in
- Statistics about a match (outcome, injuries, event data etc.)

6 Results

6.1 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.

The resulting scree plot from doing the PCA is seen in Figure 6. 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 7 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 8. 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",

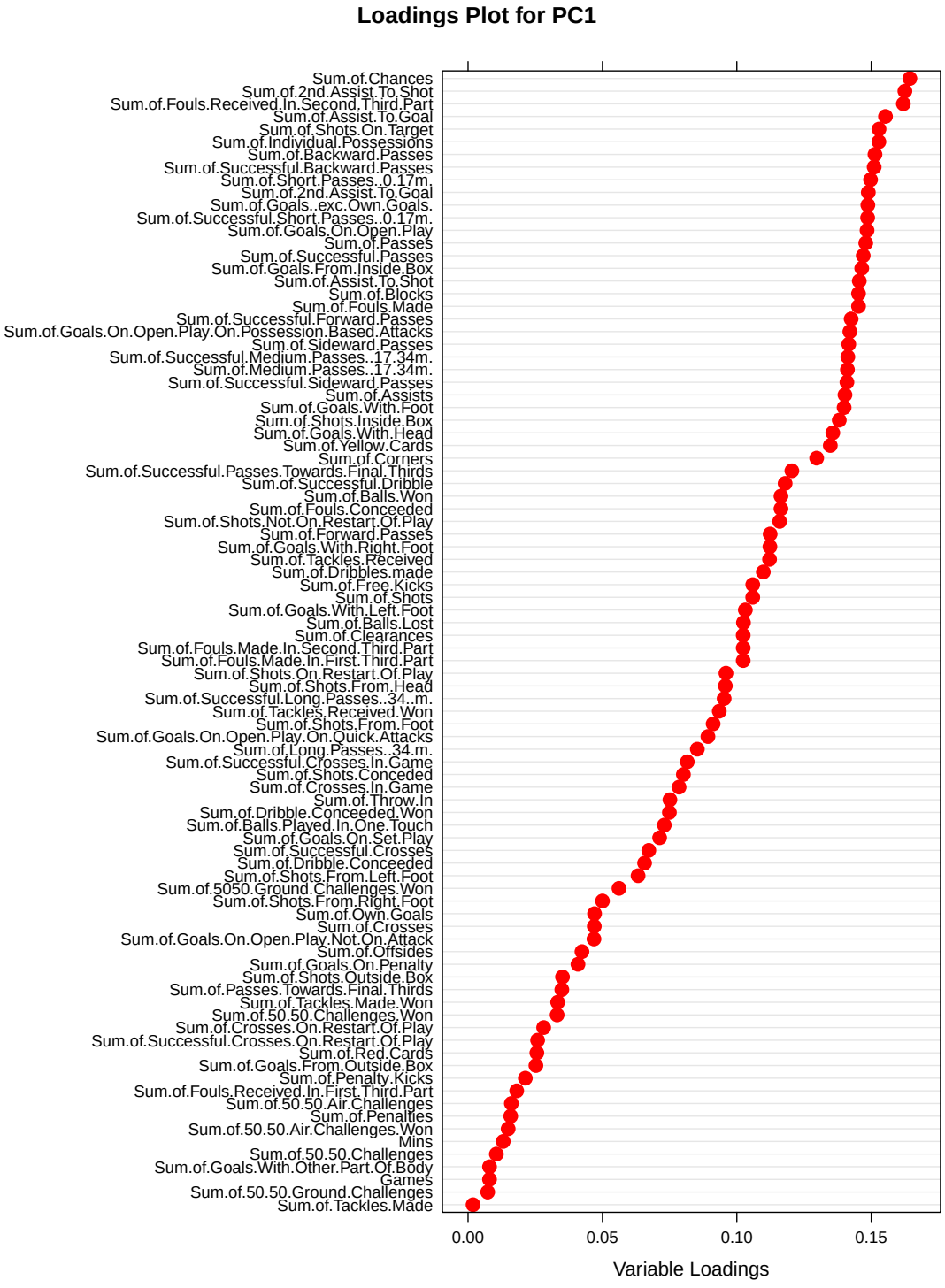


Figure 7: Loadings for PC1

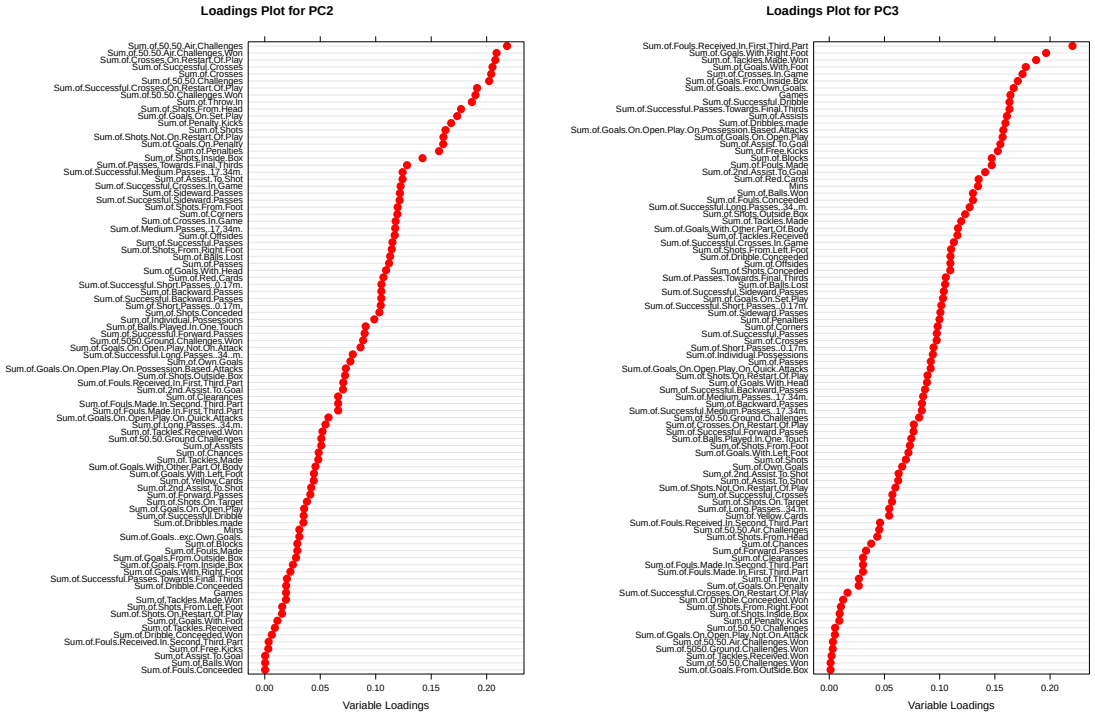


Figure 8: Loadings for PC2 and PC3

"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 9a 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.

In Figure 9b 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.

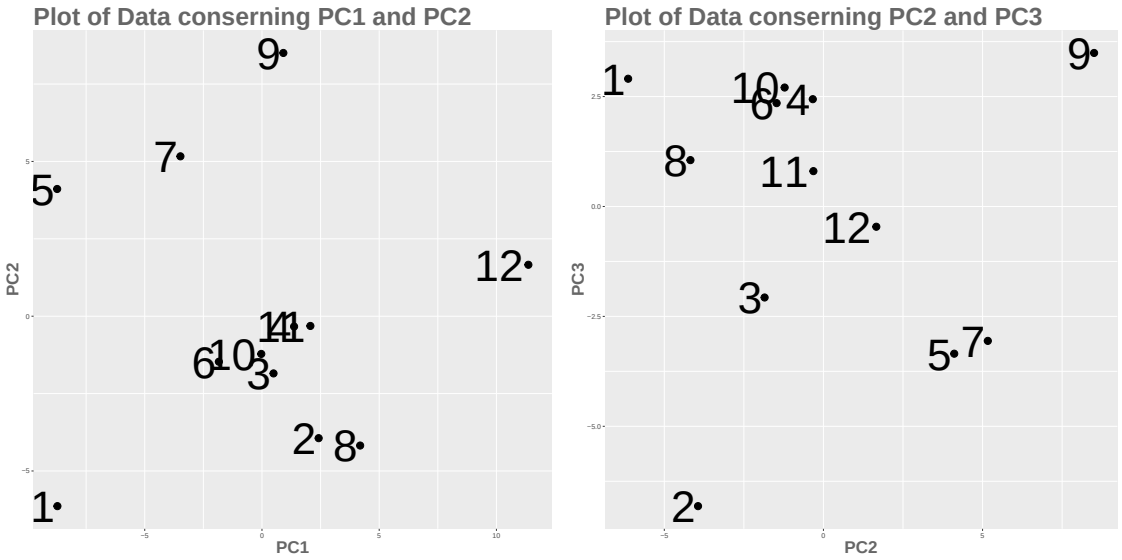
6.2 Success Rates

6.2.1 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



(a) The data plotted in regard to PC1 and PC2 (b) The data plotted in regard to PC2 and PC3

Figure 9: Data plotted in regards to principle components 1-3

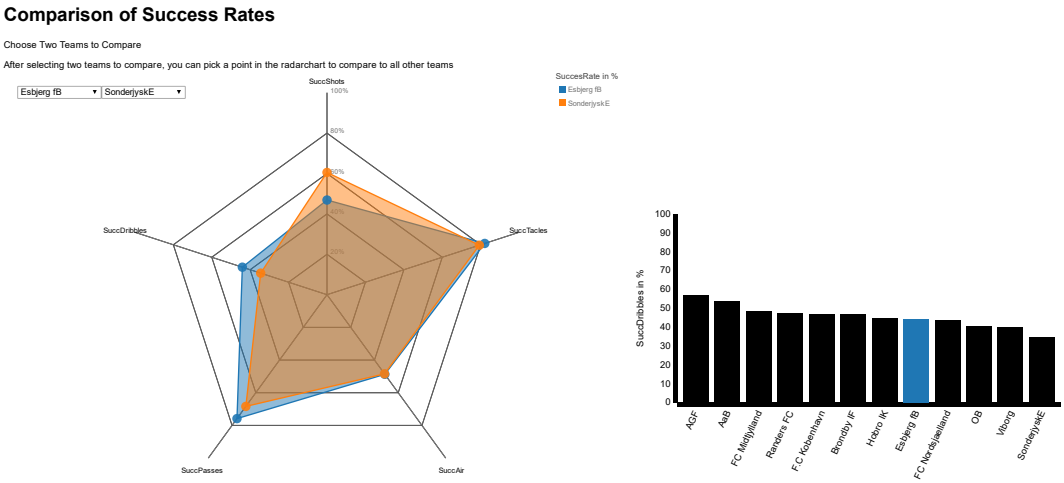


Figure 10: Idiom showing the comparison of the success rates using a radar chart for comparing two teams and bar chart for comparing with all other teams in the league

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.

6.2.2 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.

6.3 Comparison viz

What-why-how

The visualization attempts to give an overview of the three matches played between FCM and Viborg this season from FCM's point of view. The visualization presents various information regarding where touch-related data has taken place and also keeps track of the rate which different types of errors has occurred in percent.

The purpose is to allow the user to explore how a top-tier team such as FCM, deducted by the fact that they won the league in 2014/2015 and at the moment of writing is number three, can play against a new-coming underdog team from first division and in some cases win, play tie and lose. The hypothesizes is that their play-style will change when being behind, partly because they are not playing well in the first place, but also due to the pressure of losing to a on paper worse team.

To strengthen the hypothesis the visualization gives two different views where information can be deducted to find differences. The first one is a map which display where curtain events have taken place. The type of event is determined by the user in the selection-menu in the panel to the right. Six options are available and can be categorized into three overall themes which are: Aerial duals, shots and passings where for each the user can also select whether he wants to inspect the "successful" ones or the those that went wrong. By mapping the events, the user can look for clusters in the data and may find patterns to where curtain events happen. To aid in finding such clusters, two more options are available in the panel. The first one allows the user to select which matches should be inspected as the dots on the map may at time be so overwhelming that nothing can be concluded at all. Each match is represented by its own color with high variety in hue and brightness so they

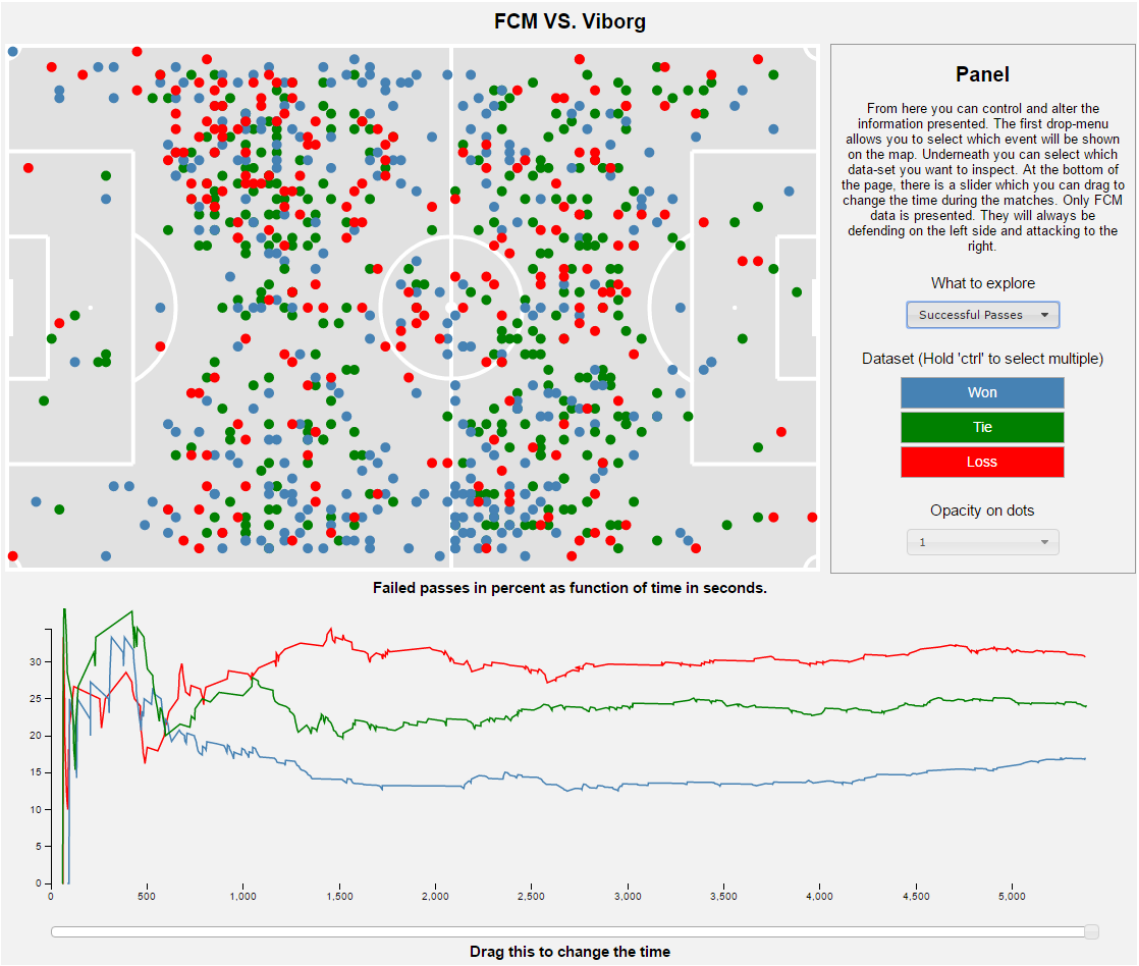


Figure 11: Kewl

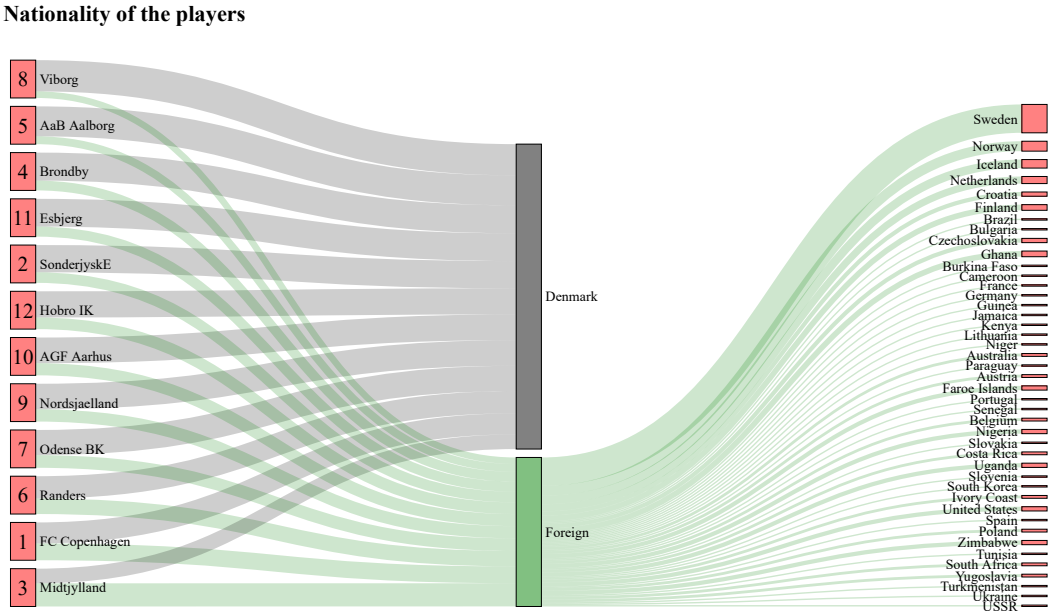


Figure 12: Sankey-Chart showing the distribution of Danish vs. foreign players on the different teams

are easier to differentiate and also the background is very neutral to avoid interference with the inspected elements. The background also have the lines of a real football field to give a better feel of where on the field the events have happened. The last option in the panel allows the user to select an opacity of the dots. As all the coordinates have been altered in such a way that it looks like FCM is always playing on the left side of the field, many dots may at times overlap and thus making them slightly more transparent enables the user to spot overlaps. The altering of the coordinates, serves the purpose of creating a more intuitive view to spot cluster and thus the opacity serves an important part in the exploration. The second view is a line-chart which illustrates the relation between two variables as a percent on the y-axis and the time which has passed in seconds on the x-axis. The relation which is shown depends on the selection in the panel, so it may be how many percent of the passes failed, shots were missed or aerial duels lost. The number lines will also toggle to the selection of matches in the panel. However, the opacity can not be change here as it is not necessary to read the information successfully and would merely be a distraction. Lastly at the very bottom is a slider which determines which point in the three matches the user desires to inspect. The slider also contains ranges so snapshots of the match can be inspected in detail.

The user is presented with two chart types, the map shows where curtain actions have played out on the map

Code

6.4 Nationality of players

6.4.1 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

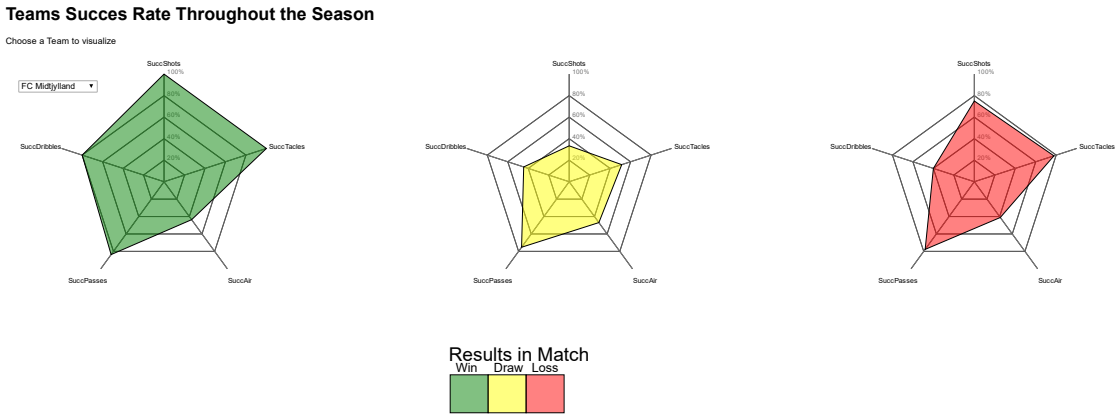


Figure 13: Comparison of a Teams success rates throughout the season, split in wins, draws and losses

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 12.

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 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.

6.4.2 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.

6.5 Success Rates Animated

6.5.1 What-why-how

Figure 13 shows the multi view Radar Chart comparing a teams success rates. Concerning the marks and channels we use position on a common scale in terms of the different axis of measure together with area, as the combined area of all the scores on each axis. The result of the match is categorised by color hue, in easy separable colours.

The action of the idiom is to compare a teams success rates through out the season in wins, losses and draws. It summarises the teams success throughout the season. One other action is enjoyment being able to look at the teams rates changing from match to match. The target is to find similarities between the different matches in each of the match results, and maybe find some common shapes for the different teams.

This is done by a idiom facet into three views. The views are linked on having the same data variables shown, but on different data. The view is manipulated by animation, looping over the different matches for the selected team. It is a smooth transaction between the matches, to create context. The raw data is reduced through aggregation by calculating new attributes.

6.5.2 Code

In R we load the data file, convert columns to be numerics, aggregate by team, calculate the success rates, transforming the data to a format suited for the radar chart, select the columns we are interested in, and write it to csv.

The D3 radar chart is based on [d3noob(2013)], and modified to become animated. We create an update function which is called continuously with different data matches to be drawn.

6.6 Custom Bubble Chart

6.6.1 What-why-how

What. The data being presented in the visualization is the statistics of the teams from the current season. The includes parameters such as their current ranking in the league and the total shots on goal, number of yellow cards etc.

Why. As the user is able to choose what is encoded in the different dimensions, the user can easily test hypotheses about correlations between different parameters, for example how rank, shots and shots on goal affect eachother. The user can also discover new correlations by simply browsing through the different parameters and trying different combinations. The visualization offers a "fun" platform where the user can play with the different parameters. The transition when the user has changed the options is animated such that the user can easily follow the circles with their eyes, and in that way see how a change in an option changes the order between the circles.

How. In the beginning of the iterative design process, the visualization started out by only having a vertical axis and size encoding. This provided meaningful information to the user, but based on feedback another axis was added such that three different parameters could be encoded in the graph. However, as the last axis was added, it turns out that the bubble chart had been reinvented. The only minor difference between this visualization idiom and the traditional bubble chart is that this visualization allows the user to interact with the visualization and change the parameters when they wish to. The time complexity of this specific implementation is O(nlgn) where n is the amount of different parameters available to be chosen. The reason being that when a user has selected the options, the coordinates and sizes of the circles are calculated in the following way: First they are sorted based on the first parameter, and then the radius of the circle is assigned in the order that they are now in. Then they are sorted based on the second parameter, and the y coordinate is assigned based on the radius and the order they are in (we have to base it on the radius as well such that there is equal distance between them and no overlaps). Finally they are

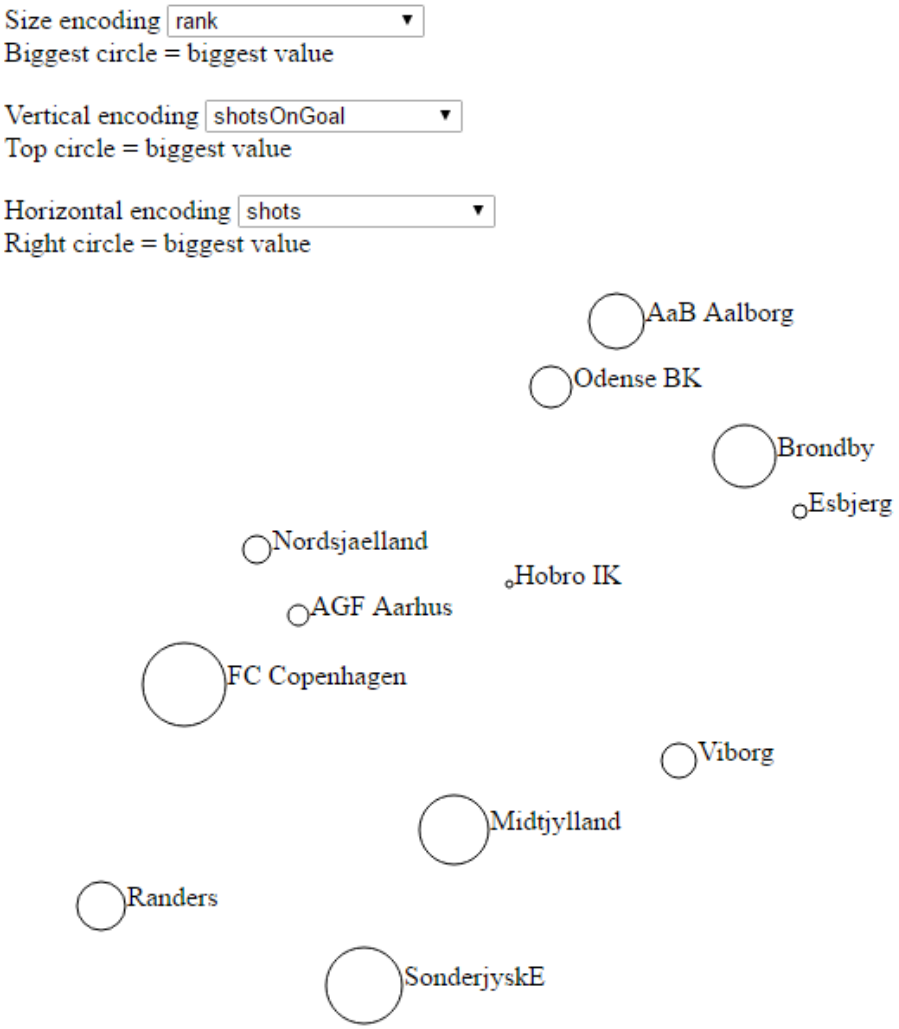


Figure 14: Visualization of statistics of teams from the current season. Encodes rank, shots and shots on goal

sorted based on the third parameter, and the x coordinate is assigned. Since we sort 3 times, the complexity is going to be O(nlgn), assuming that we can sort in O(nlgn) time. While there might be smarter approaches than this very straightforward approach, it is not of any concern at the moment, as the different number of parameters a user can choose in football data is very limited. The smart thing about the implementation is that it is very flexible. All of the data that the user is being presented with is based on the data that is in the data file that the algorithm imports. The data file that is supplied should be encoded in JSON as a list of objects, where all of the objects have the same attributes, and all of the values are numerical except for the "name" attribute which is excluded from the drop down list, and is used as a label on the circles. This means that the implementation can be used to visualize many different things. It automatically extracts the attributes that must go in the drop down list, and assigns the names to the circles. The mark chosen in this visualization is a point, and the visual channel chosen to encode data into the mark is size and position. Color could have been chosen aswell, and a fourth parameter could be encoded within that, but for football data purposes, 3 dimensions suffices. The data that the visualization uses is preproccessed in R and exported in the format that was described above.

6.6.2 Code

This is the pseudocode for the way that the x, y and size dimensions are calculated for each circle. Since we receive the data as a list of objects, we can append/change the x, y and radius of each circle directly on the object as each circle represents an object. The updateDimensions function takes as arguments the list of objects, the parameters that it should encode the circle by and the distance between each circle. Note that when calculating the y coordinate of the circles, we sort the objects in reverse order as coordinates in graphical spaces usually have the origin in the upper left corner, meaning that it is counter intuitive for users compared to how they normally view graphs, meaning that we have to inverse the y-coordinates as to not make it confusing. After the dimensions have been updated, the circles would transition into their new dimensions (i.e. grow or shrink and move along the axes).

```
updateDimensions(objects, sizePar, xPar, yPar, distance) {
  sortByParameter(objects, sizePar)

  currentRadius = 1

  for(object in objects)
    object.radius = currentRadius
    currentRadius++

  sortByParameterReverse(objects, yPar)

  prevRadius = 0
  prevY = 0

  for(object in objects)
    object.y = prevY + prevRadius + distance + object.radius
    prevY = object.y
    prevRadius = object.radius

  sortByParameter(objects, xPar)
```

```
21
22    prevRadius = 0
23    prevX = 0
24
25    for(object in objects)
26      object.x = prevX + prevRadius + distance + object.radius
27      prevX = object.x
28      prevRadius = object.radius
29  }
```

6.7 Field Events

6.7.1 What-why-how

What. The data being presented in the visualization is event data from a football season. Event data is things such as a good pass to another player or a goal attempt. From the API we can request events from a match. The event data include the type of event, the time stamp and the coordinates. What is being shown in the visualization is then all of the event data from an entire football season aggregated such that when a user selects good passes and the time interval 0 to 10 minutes, the user will be presented with all of the good passes that has happened in the season during the first 10 minutes.

Why. This visualization could be categorized as being in the exploratory data analysis category, as it allows a user to browse through the data in a very intuitive way, such that the user can get a feeling of how common each event is, and where on the field it is often carried out. But the visualization can also be used if the user have a specific hypothesis that they want to test. The visualization was first build because we had a hypothesis about the goal attempts, where the hypothesis was about how football players might take more risky attempts at goals during the late part of a game, as they perhaps are more desperate to score. To test this specific hypothesis, a user can select goal attempts and then use the slider to see how the goal attempts evolve throughout a match. The result of the hypothesis test is described later in the report.

How. As the data to be shown is spatial and discrete data, it is being presented in a field (which happens to be a football field). The type of mark being used is a point and the visual channel used to encode the mark is spatial position in both vertical and horizontal direction. To control what kind of event a point represents, the user is presented with a drop down menu, where the user can select the type of event. An alternative to this option could be to use the visual channel color, such that goal attempts was blue, good passes was yellow and so on. However, it was decided that only one type of event should be shown at once, as multiple types of events at once could quickly become noisy. Alternatively, a user could select two types and choose the color of them.

Because the resolution of the coordinates is not very high (coordinates are seperated by 1 meter), a lot of points are going to overlap because a lot of events happen at the same place. To avoid that this is going to disturb the meaning of the data, two things has been done. The first thing is that the opacity of each point is lowered a little, meaning that points are actually a bit transparent. This means that if two points are on top of eachother, the point will appear more opaque. The other thing that is done is that there is added or subtracted a small random amount from each point's x and y coordinates. This approach is called "jigger" and is often used when overlapping data are to be visually

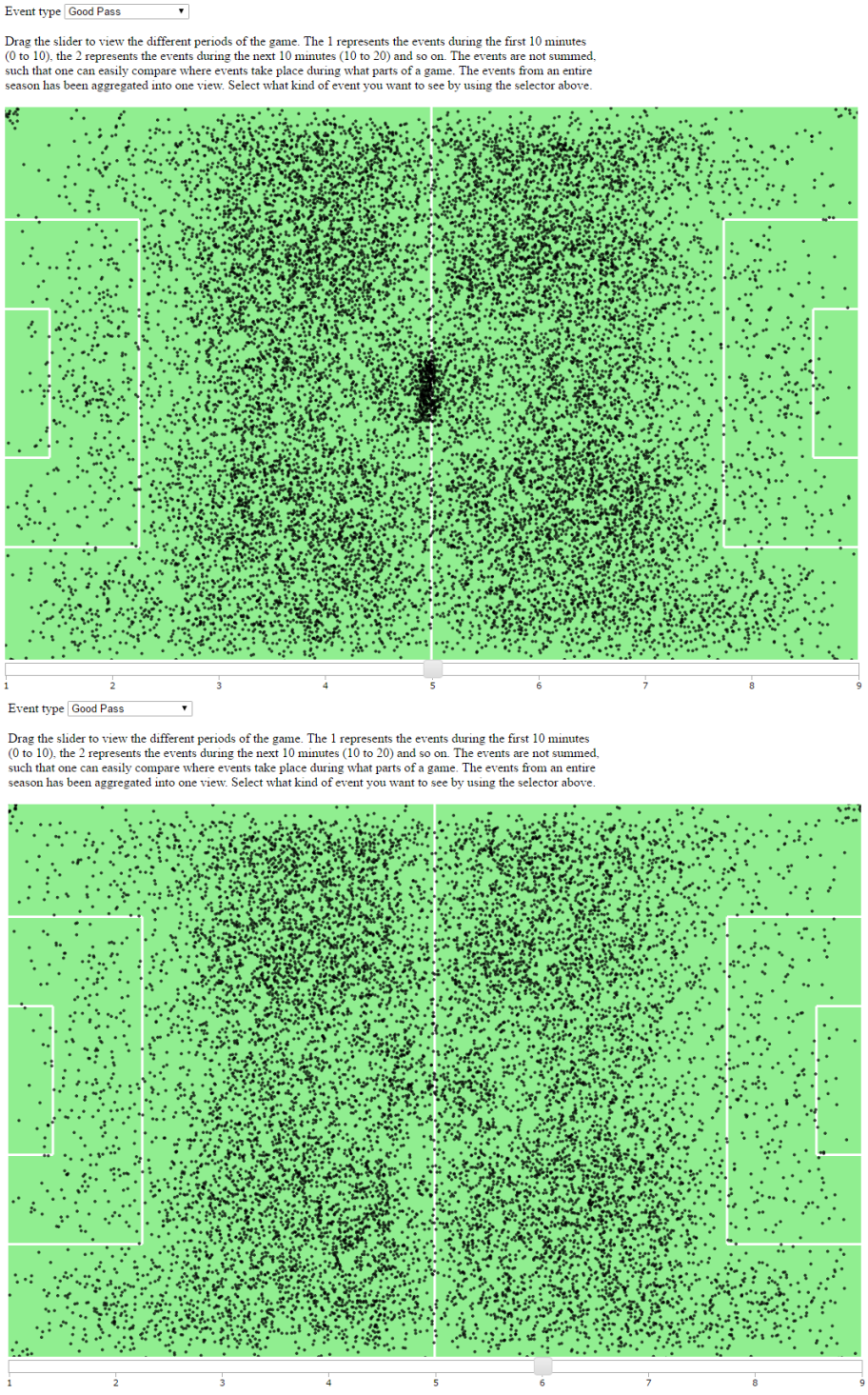


Figure 15: Visualization of event data in intervals during a match. Good passes during the 40 to 50 minutes compared to 50 to 60

presented without overlapping. However, there might still be overlaps, as it is actually not checked whether or not the new x and y coordinates of a point collides with another point. One could make an implementation where this is checked, but even if a collision check is done by an array look-up in O(1) time, an implementation could easily result in a worst case time complexity of O(n) where n is the number of events, if the events are laid out in an unfortunate way (for example if they all have the same x and y coordinates, and they are centered around those coordinates, and the algorithm checks for a place to place a point in a circular fashion centered around the original coordinates). The time complexity of the algorithm behind the visualization is O(n), where n is the amount of data points, as the data is read from disk, then the data is binded to the points, and the points' coordinates are updated. Finally, the old data points are removed. This time complexity assumes that the D3 library data binding operation is done in O(n) time. Note that this is not that good of a time complexity for a dynamic visualization, and sometimes it can be slow depending on how fast the data is loaded from disk or whether it is already cached or not. Alternatively the different events and periods could be preprocessed even further and saved as images that is displayed instead, but this would still require loading the images from the disk, but would make each change of visualization O(1) instead of (n). This approach would however limit the options of what can be done with the data in D3, if one wishes to develop the visualization further.

The data is prepared in R before it is used by D3. The pseudocode for the R code can be seen in the code section. The API that we get our data from can only serve event data from one match at a time. This means that we have to automatically query the API for the event data from all of the different events that is in a season, and then aggregate all of the data into one data frame. From that data frame, the data is sorted into different data frames depending on their event type. The data frames that only contain one event type is then split into 9 different data frames depending on their time in the match. The events during minute 0 to 10 is split into one data frame, the events during 10 to 20 is split into another and so on. These data frames are then saved to seperate files on the disk, where the name of the file is (type of event) + (period in the game) + (.json). 171 different files are produced from this process, which shows that it is important to automate procceses as this. These files are then placed in a directory where the javascript of the visualization can read them. Because we label the event types the same way as the API does, we have a global naming scheme, meaning that the javascript can simply reconstruct the file names and request them from the OS. Another benefit of doing this automatically is that the visualization can quickly be adopted to different kinds of data, and show different seasons etc.

6.7.2 Code

The visualization is made using the D3 library, but the visualization itself is coded entirely from scratch, and is not just a modified example (except for the slider).

The pseudocode for the data processing in R is as follow. Where eventID is the id of a match and touch means an event within the game, for example a good pass.

```
1 getTouchesAndWriteToFiles() {
2   teams = getDataFrame(teams)
3
4   eventIDs = new DataFrame()
5
6   for(teamID in teams.teamID)
7     events = getDataFrame(events + teamID)
```

```
8     eventIDs = combine(eventIDs, events)
9
10  eventIDs = distinct(eventIDs)
11
12  allTouches = new DataFrame()
13
14  for(eventID in eventIDs.IDs)
15    touches = getDataFrame(touches + eventID)
16    allTouches = combine(allTouches, touches)
17
18  splittedTouches = split(allTouches, touchID)
19
20  for(touchDataFrame in splittedTouches)
21    for(i = 0 to 8)
22      currentTouches = select(touchDataFrame, minutes >= i*10 AND minutes < (i +
          1)*10)
23      toWrite = toJSON(currentTouches)
24      write(toWrite, touchDataFrame.name + i + ".json")
25  }
```

7 Discussion

8 Conclusion

9 Usability

10 Bibliography

[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;.

[Seltman(2016)] Howard Seltman. Experimental design for behavioral and social sciences, 2016.

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

[Weimer(2016)] Paula Weimer. History of statistics and probability, john wilder tukey, 2016.

11 Appendix

12 Process Evaluation