

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

29th May 2016

1 Abstract

2 Preface

Contents

1	Abstract	2
2	Preface	2
3	Introduction	5
4	Theory	5
4.1	Design Process	5
4.2	Design	6
4.3	Data Types and Data Sets	7
4.4	Idioms	8
4.4.1	Marks and Channels	8
4.4.2	Graphs	9
4.5	Visualization validation	9
4.5.1	Four levels of visualization design	9
4.5.2	Visualization validation	10
4.6	Facets and View Manipulation	10
4.6.1	View Manipulation	11
4.6.2	Facet	12
4.7	Exploratory Data Analysis	13
4.7.1	Principle Components Analysis	14
4.8	Tools and Technologies	14
4.8.1	R	14
4.8.2	D3.js	17
5	Data and API	18
6	Results	18
6.1	Exploratory Data Analysis	18
6.1.1	Principle Component Analysis	19
6.2	Source of Goals	21
6.2.1	What-why-how	21
6.2.2	Code	22
6.3	Passes -Direction	23
6.3.1	What-why-how	23
6.3.2	Code	24
6.4	Passes -Length	24
6.4.1	What-why-how	24
6.4.2	Code	25
6.5	Shots -Head vs Foot	26
6.5.1	What-why-how	26
6.5.2	Code	27
6.6	Shots -Inside box vs Outside box	27
6.6.1	What-why-how	28
6.6.2	Code	28
6.7	Success Rates	28
6.7.1	What-why-how	28
6.7.2	Code	29

6.8	Season Review FCK	29
6.8.1	What-why-how	29
6.8.2	Code	30
6.9	Fouls Committed over season	31
6.9.1	What-why-how	31
6.9.2	Code	31
6.10	Comparison viz	31
6.11	Nationality of players	33
6.11.1	What-why-how	33
6.11.2	Code	34
6.12	Success Rates Animated	34
6.12.1	What-why-how	34
6.12.2	Code	35
6.13	Custom Bubble Chart	35
6.13.1	What-why-how	35
6.13.2	Code	37
6.14	Field Events	37
6.14.1	What-why-how	37
6.14.2	Code	40
7	Discussion	40
8	Conclusion	43
9	Usability	43
10	Bibliography	45

3 Introduction

In modern day society, 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 visualizations. Visualization of data 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, there are several considerations the designer must make, including: What the semantics are of the data being visualized, what actions should be made to reach the visualization, what is the target of the visualization and how the visualization is going to be created. It is also 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. Examples of this include limited variables in the visualizations and contrasting colours. Given that visualizations are created to simplify large and complex data sets, sacrifices must be made to get the result across to humans. Too complex visualizations will not complete the task of presenting the data in an easily understandable manner, as human's won't be able to process all of the information. In order to do this, we will apply principles from the field of visualization to present football data. We will use tools such as the programming language R to process data and plot static visualizations, and use the JavaScript library D3.js 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 understanding the type of data that is to be visualized, considering how to filter the data to get the relevant data for the individual visualizations and how to create the visualization. We will discuss the results of the visualizations, specifically what we can and can't learn from them, and reflect on how these visualizations could be used in other fields.

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

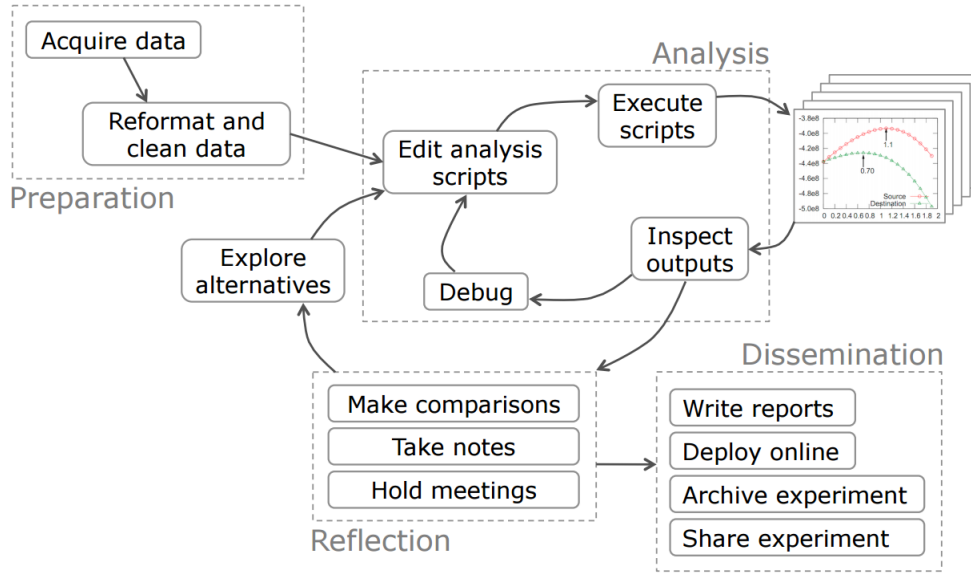


Figure 1: The model showing the iterative process [Guo(2012), Chapter 2]

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]

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 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

In the world of data visualization, there are many different ways of presenting data. In fact, there are so many ways, or idioms, that it is impossible to comprehend the (dis)advantages of every single one. These idioms range from simple static idioms, such as bar charts, scatter plots and line charts, to more complex ones, such as static idioms linked together by interaction, such as the comparison of success rates, described later.

When selecting an idiom to represent a dataset, it is important to know the strengths and weaknesses associated with that idiom. Different data sets require different idioms, and in order to choose the right one, it is paramount to know about marks and channels.

4.4.1 Marks and Channels

Marks are basic graphic elements in a picture, while channels control the appearance of the marks [Munzner and Maguire(2015), Chapter 5, p. 95-96]. In this respect, marks can be anything from small points in a scatter plot, to lines in a line chart, or even areas. Channels can be the position of the mark, the shape, tilt, length or color of the mark, or anything else that alters the appearance of the mark. In a bar chart, the mark is typically a line with the channels being the length of that line, as well as horizontal categorical information, for example the different teams in our grouped bar charts. Color can also be used as a channel, making it possible to encode more information in the chart, for example by plotting different variables in a group, each variable having its own color.

Not all channels can be used with all marks. If a two-dimensional mark is used (area), it is not practical to use a shape-based visual channel, as it makes it harder to compare the different marks with one another (comparing the area of a triangle to the area of a rectangle, for instance). Therefore, if one runs out of channels to encode information with, it may be wise to contemplate using other kinds of marks, or even other visual channels, if the usage of one channel means that other channels can't be used.

The most effective channels should be matched with the most important attributes, in order to increase perception of the visualization, and therefore efficiency of the idiom. When the most important attributes are the easiest to compare, it strengthens the visualization as a whole. The most effective magnitude channel (channels that describe quantity) is aligned spatial position, with unaligned spatial position channels being second best [Munzner and Maguire(2015), Chapter 5, p. 101]. After these comes length, angle and area, length being more effective than area to convey quantitative data. The most effective

identity channels (channels that describe categories) are spatial regions, followed by color hue.

4.4.2 Graphs

- **Scatter Plots**

Scatter plots are effective when visualizing few (typically two) variables of a large dataset [Iliinsky and Steele(2011), Chapter 5, p. 54]. Visual channels typically used in scatter plots include horizontal and vertical spatial position, color hue, area, as well as shape. However, if area is used on the marks, one should not use shape as an identity channel, but instead color hue. Scatter plots can be used to quickly gain an overview of the data set, since it is possible to plot large quantities of observations. In our ‘Comparison viz’ and our ‘Field Events’, a scatter plot is utilized to show the x and y coordinates of different events, and color hue is used in the former, to show if the match was won, tied or lost.

- **Line charts**

Line charts are good for displaying continuous data or trends, and they can be much cleaner to look at, than bar charts. They usually have independent data on the x-axis, such as time or matches played, and dependent data on the y-axis, such as points in the league.

- **Bar charts** Bar charts are widely used for comparing variables, either across, or within categories. Later it is described how we utilize grouped bar charts to compare variables across teams. Groups are better than stacked bars, since stacked bars don’t share the same baseline on the top bars.

- **Radar Chart** Radar charts are useful for comparing independent variables, such as the different success rates between teams. They can be used to show up to seven variables simultaneously and efficiently, quickly creating an overview of the differences in observations. For comparing dependent variables, such as a distribution, a histogram or a pie chart should be considered instead, as well as other idioms.

4.5 Visualization validation

This section briefly summarizes some of the considerations / steps that should be taken when validating that visualizations show what is intended.

4.5.1 Four levels of visualization design

Visualization design consist of four nested levels [Munzner and Maguire(2015), 68]:

1. Domain situation, where it is considered whether or not a problem could be solved or even partially solved by visualization
2. Data/task abstraction, where the data that contain the (partial) solution is found, and abstracted from the domain.
3. Visual encoding/interaction idiom, where it is considered how to visualize this data.
4. The algorithm level, where the algorithms behind the visualization idioms are constructed.

Splitting visualizations up in four parts like this, will make things easier when analysing, and if something doesn't work, finding the root of the problem will be easier as well. There are two ways of going through this model, first being down-stream, where you, the designer try to solve some visualization problem that is there for a group of people. In this case, there are usually already visualizations that will suit your need. The other is up-stream, where the objective usually is to create some new visualization, starting at the algorithm, or idiom step, working up to test practical use of the visualization.

4.5.2 Visualization validation

When a visualization is made, it is time to validate it. There are several threats to a visualizations validity, neatly summarized in Visualization analysis and Design [Munzner and Maguire(2015), 75]

- Wrong problem: You misunderstood their needs.
- Wrong abstraction: You're showing them the wrong thing.
- Wrong idiom: The way you show them doesn't work.
- Wrong algorithm: Your code is too slow.

Wrong problem: this is most likely to happen with a down-stream design, where there is some sort of miscommunication between designer and the people who need the visualization. This will almost guaranteed cause problems in the data abstraction step as well, so if something seems to be wrong there, it would be a good idea to check the problem step as well.

To solve this, one should first contact and interview the target group if possible, however if the target group is too broad, or very general they might not know what they need themselves, in which case validating everything else first, and then looking at adoption rates would be a good idea.

Wrong abstraction: This occurs if the data selected simply does not have what is needed to solve given problem. A common way to validate the correctness of the abstraction, is getting target users to use the visualization system, and see if the user(s) find it useful.

Wrong idiom: This happens if the visualization technique isn't fit for the given dataset or simply doesn't lower the cognitive load for the user over looking at the raw data. If something is wrong at this level, using a different visualization technique should be considered if the designer is working down-stream, otherwise sitting down and double-checking the visual encoding / interaction, followed by getting people to test the visualization for different purposes, and adjusting based on their feedback.

Wrong algorithm: The major problem here is, if it takes seconds, minutes or even more, where a user would expect a results in less than a second.

A good first step, is to analyse the complexity of the algorithm, and see if any apparent improvements can be made. After that, it is worth to take a look at the input data, and see if it can be pre-processed further, to increase the visualization loading times even further. Then there are other cases, where the problem is in fact, the memory usage. In these cases it should be considered to reduce memory usage where possible, even if it will reduce performance.

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]

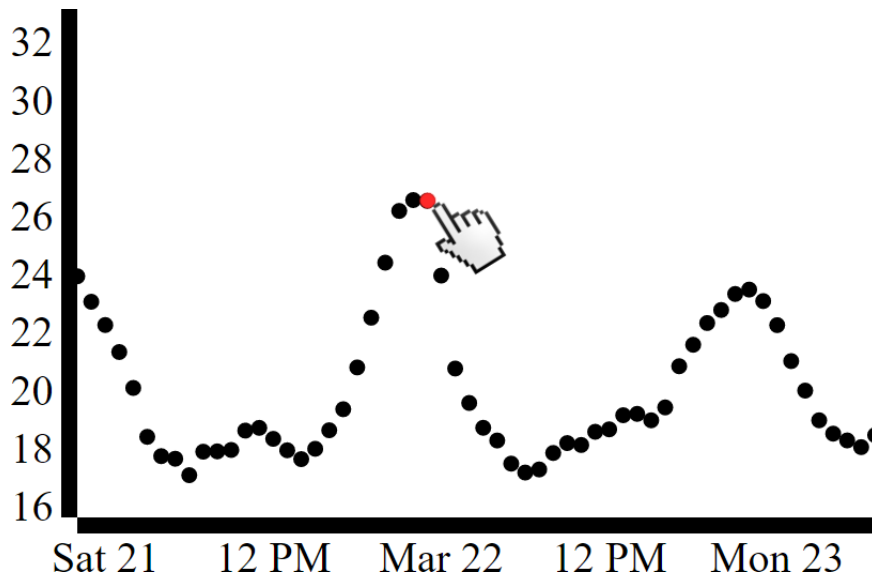


Figure 2: Highlighting the selected element

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 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

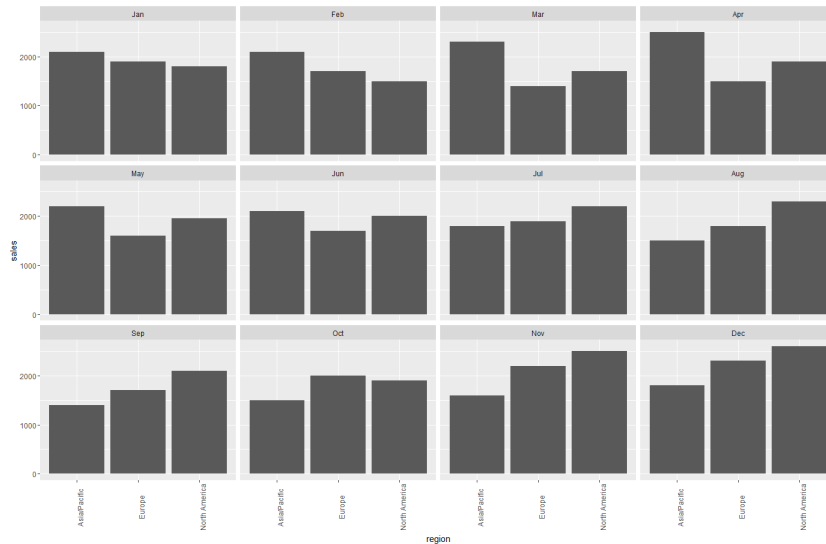


Figure 3: An example of small multiples

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. 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.

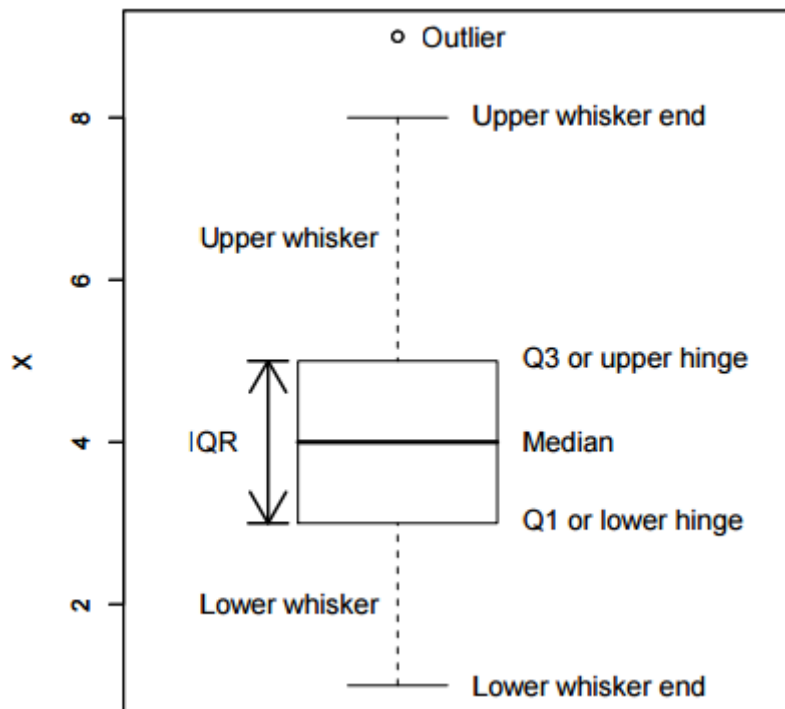


Figure 4: A boxplot[Seltman(2016)]

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 relative value of IQR, the more the data is spread, and the smaller

the relative value of 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 each other, and very quickly be able to recognize the differences between the data sets, as data has now been encoded 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 natural consequence.

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 losing 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 your 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 6.1.1 we do principle components analysis on some of our own data.

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

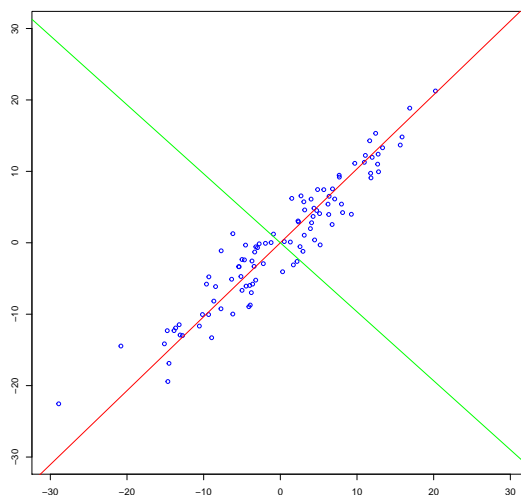


Figure 5: Some data set plotted, red line is PC1, green is PC2

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 and 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 and 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 and 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 build the dataset. When a data frame is constructed, the user often want to restructure it. 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 two 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 [Simmering(2013)] 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

D3, short for Data-Driven Documents, is a open-source JavaScript library that allows DOM (Document Object Model) manipulation based on data.

It is designed to work on most "*modern*" internet browsers, anything newer than Internet Explorer 8.

Like *R*, the code written using D3 will provide high code reusability, as different visualizations will likely include similar elements. Once a visualization is made, it can be kept up to date simply by updating the data files, or not necessary at all if data is gathered through an API.

D3 assists with binding arbitrary data to selected DOM elements.

Upon using the `data.enter()` on a selection, any DOM elements in that selection will get one element from **data** assigned to it, and if there are more elements in **data**, than in the DOM selection, elements are created for each remaining element in **data**. If on the other hand, there are more DOM elements in the selection, than in the **data**, the `data.exit()` method combined with `.remove()`, will remove excess elements.[Bostock(2016)]

Although D3 can be used to create HTML tables etc. the most common use of D3 for visualization purposes, is through SVG (Scalable Vector Graphics). SVG is based on XML, and can be modified / styled on the go through JavaScript for editing the SVG elements, and CSS for styling them. The CSS styling can be set and/or modified at runtime through JavaScript as well. Because of this, the programmer have the possibility to create visualizations that react to what the user is doing in interesting ways, or create dynamic visualizations such as time lapse bubble-charts.

Another reason to use SVG over raster image formats, is that it will look as intended no matter how much it is zoomed / scaled, where raster images like png or jpeg become pixelated when scaled. One of the major features in d3 is the Scales sub-library, which provides functionality that can transform all data given to fit inside the given range, while still maintaining the relative position / size between elements. This is especially useful when designing responsive web pages, where dimensions of the SVG element may differ.

D3's SVG sub-library brings generators for several non-standard shapes in SVG. These generators can generate SVG paths which resemble the shapes. Some of these shapes are: lines, arcs and chords. These shapes be modified with options such as interpolation, which can be used to smoothen lines in a line chart, while they are still styleable through CSS.

While this way of doing visualizations in a lot of cases is preferred due to reusability, and user interactivity, working with large datasets can give unwanted results. If the user have to load more than a few megabytes of data in order to create the visualization, it is going to take some time, and if that data results in a large amount of SVG elements, a dynamic or interactive visualization could become unresponsive, which is undesired, as users are known to be impatient.

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 convenient to work with it in R, as one don’t have to do any 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 append arguments, 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 in the response. 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 can be 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 of 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 the responses, 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
- 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 Exploratory Data Analysis

To easily see how the goal attempts change throughout a match, the goal attempts of an entire season has been aggregated and then split into intervals such that it is possible to

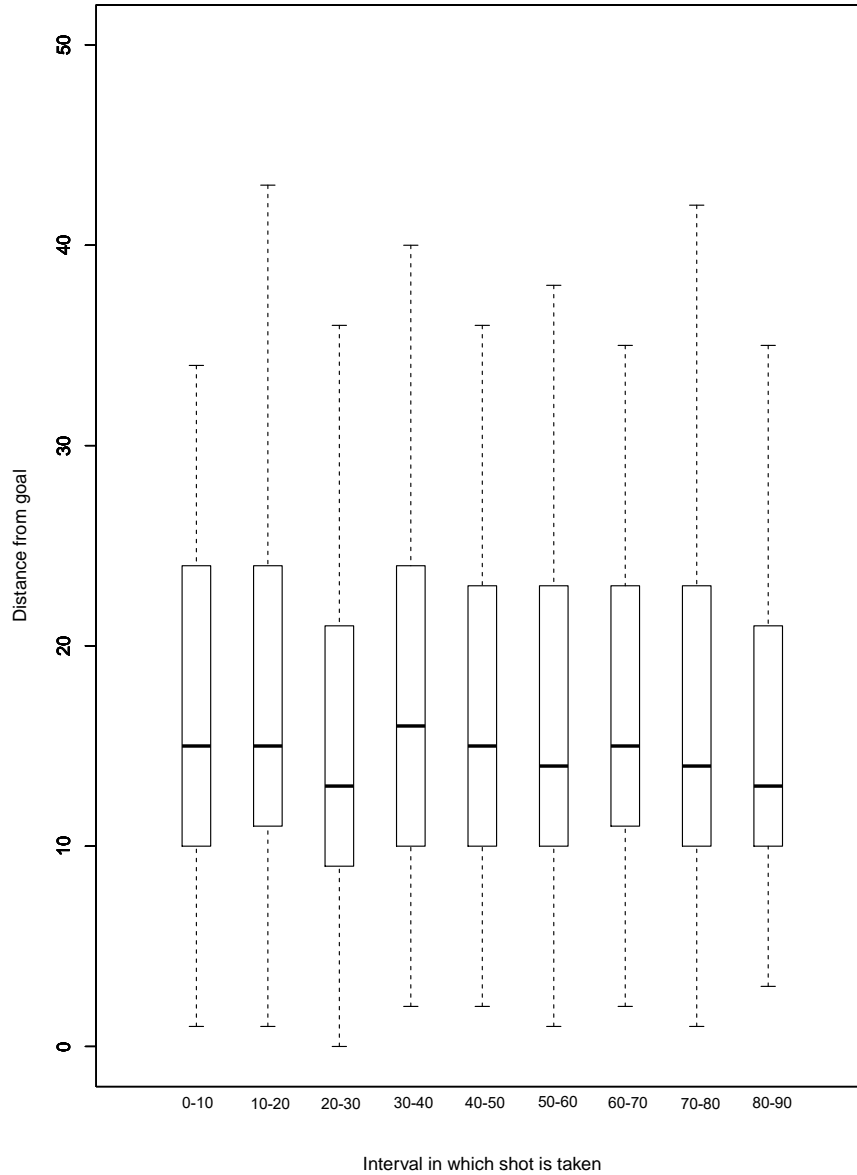


Figure 6: Box plot of goal attempts in the first 10 minutes, the next 20 minutes etc. The distance is the distance from the goal on the x-axis

see the distance from the goal a goal attempt is made during the first 10 minutes, the next 10 minutes and so on. This information has been visualized using a box plot (figure 6). As it is described in the theory section, a box plot quickly allows a user to see different things about a data set, and therefore functions as a quick way to propose hypotheses about the data.

6.1.1 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 7. The scree plot tells

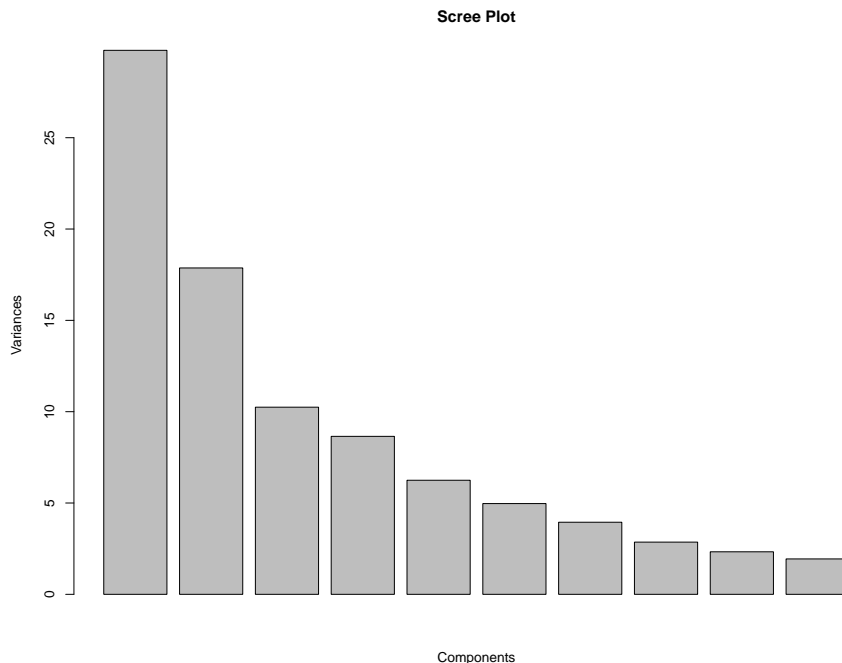


Figure 7: The resulting scree plot

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 8 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 9. 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 10a 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 10b 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.

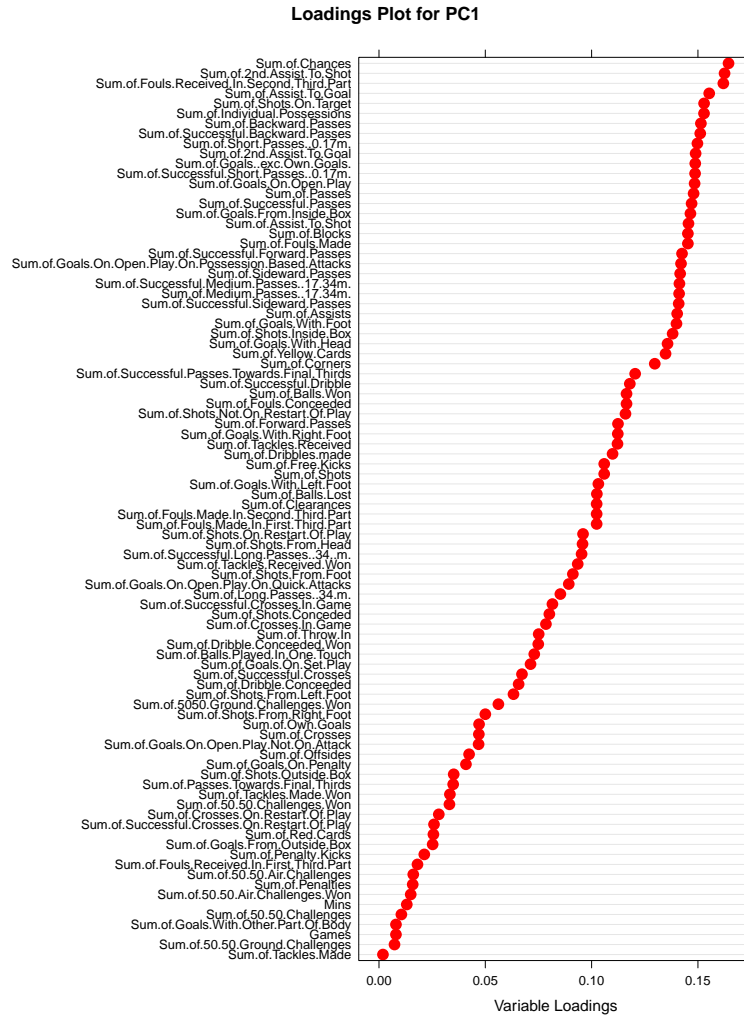


Figure 8: Loadings for PC1

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

6.2 Source of Goals

6.2.1 What-why-how

This visualization shows the average sources of goals for every team in the league. Goals are divided into five categories - Goals from not attacking open plays, penalties, goals from possession based attacks, goals from quick attacks, and goals from set plays. A grouped bar chart is utilized to clearly show the differences between the teams. Color hue determines the source of goal, and all values are in percentages. A team's percentages add up to 100, and the bar groups are ordered, based on the team's rank in the league. The purpose of this idiom is to see if there are any noticeable correlations between a team's position in the league, and how they score their goals on average.

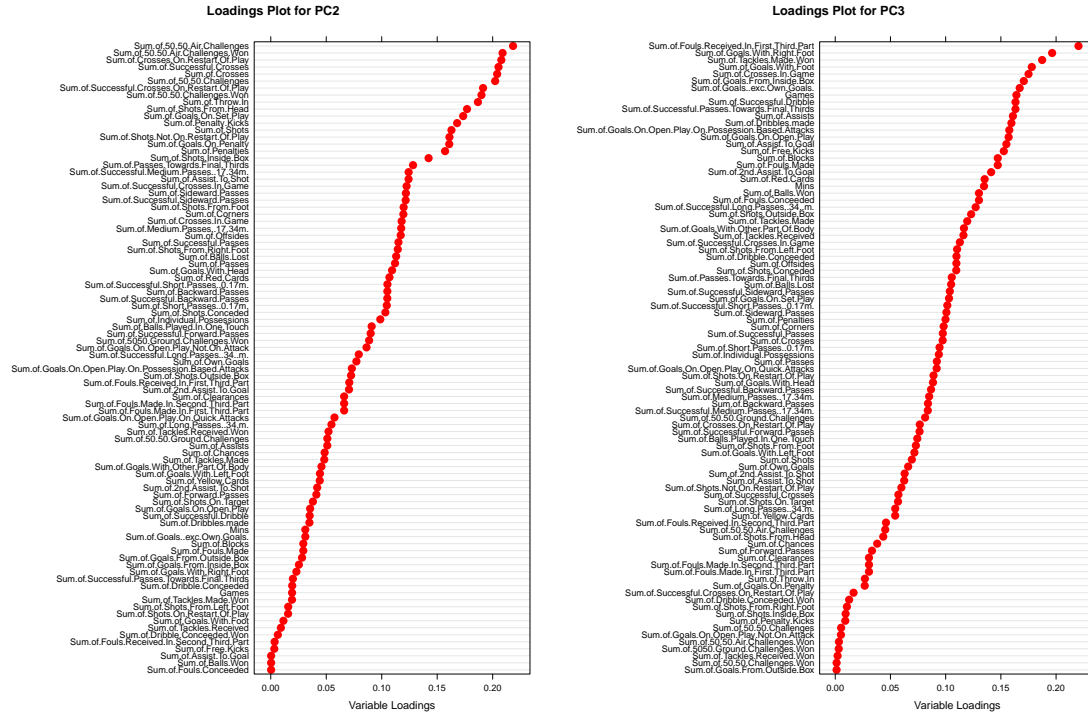
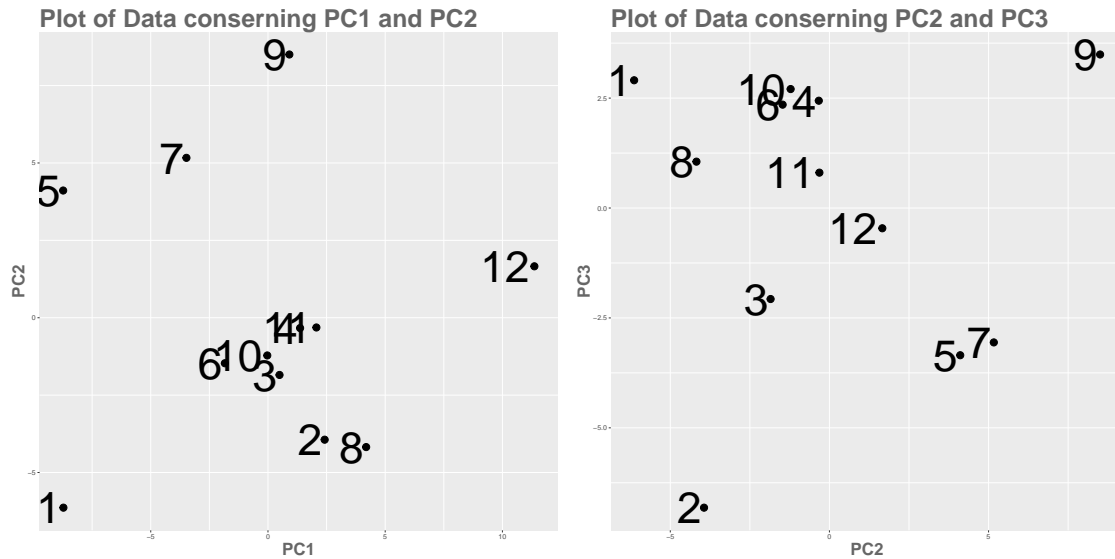


Figure 9: Loadings for PC2 and PC3



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

Figure 10: Data plotted in regards to principle components 1-3

6.2.2 Code

R was used to prepare the data from the excel sheet. The variables in the bar chart were selected from the dataset, then summed for each team. After this, the variables were divided with the total number of goals per team, to get the average distribution of goal sources. This was done in a loop, adding variables, variable names and a team's ranking to predefined lists, and then creating a data frame from these lists. It is more efficient to

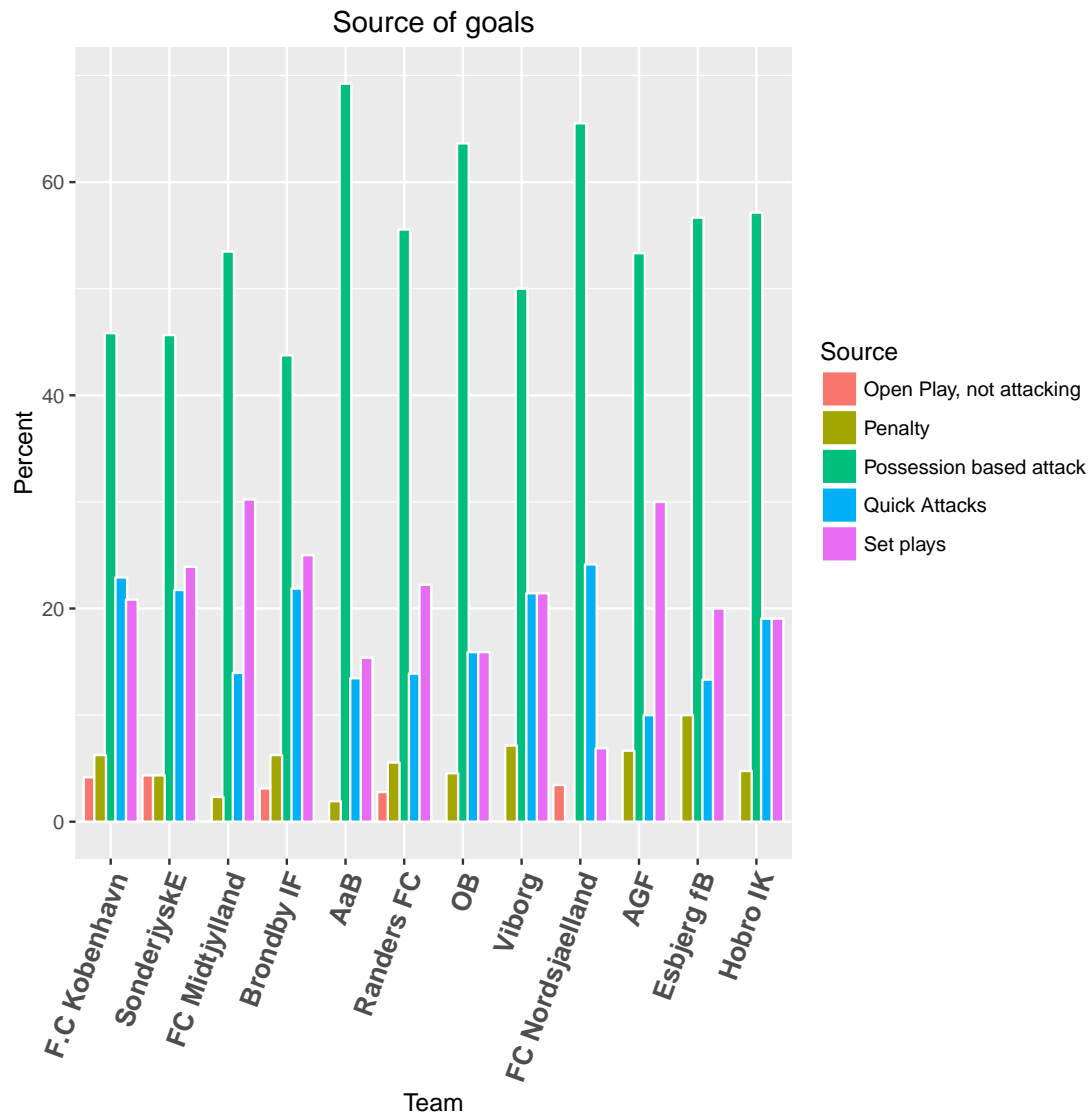


Figure 11: Grouped bar chart showing the kinds of attack leading to goals

create a data frame this way, instead of growing a predefined data frame inside the loop. The order of the groups in the bar chart is determined by the order of the levels inside the data frame, so these levels had to be reordered based on the team's ranking. The chart was then made with `ggplot2`, with teamnames on the x-axis, the percentages on the y-axis, and with the different variables as fill (Set plays etc.) Grouping of the bar chart was set with `position=position_dodge`.

6.3 Passes -Direction

6.3.1 What-why-how

The purpose of this idiom is to see if there is a correlation between the direction of a team's passes on average, and their ranking in the league. This is done by creating a grouped bar chart, with a group per team. The bar chart is ordered by the ranking of the team, with the highest ranking team as the first group from the left.

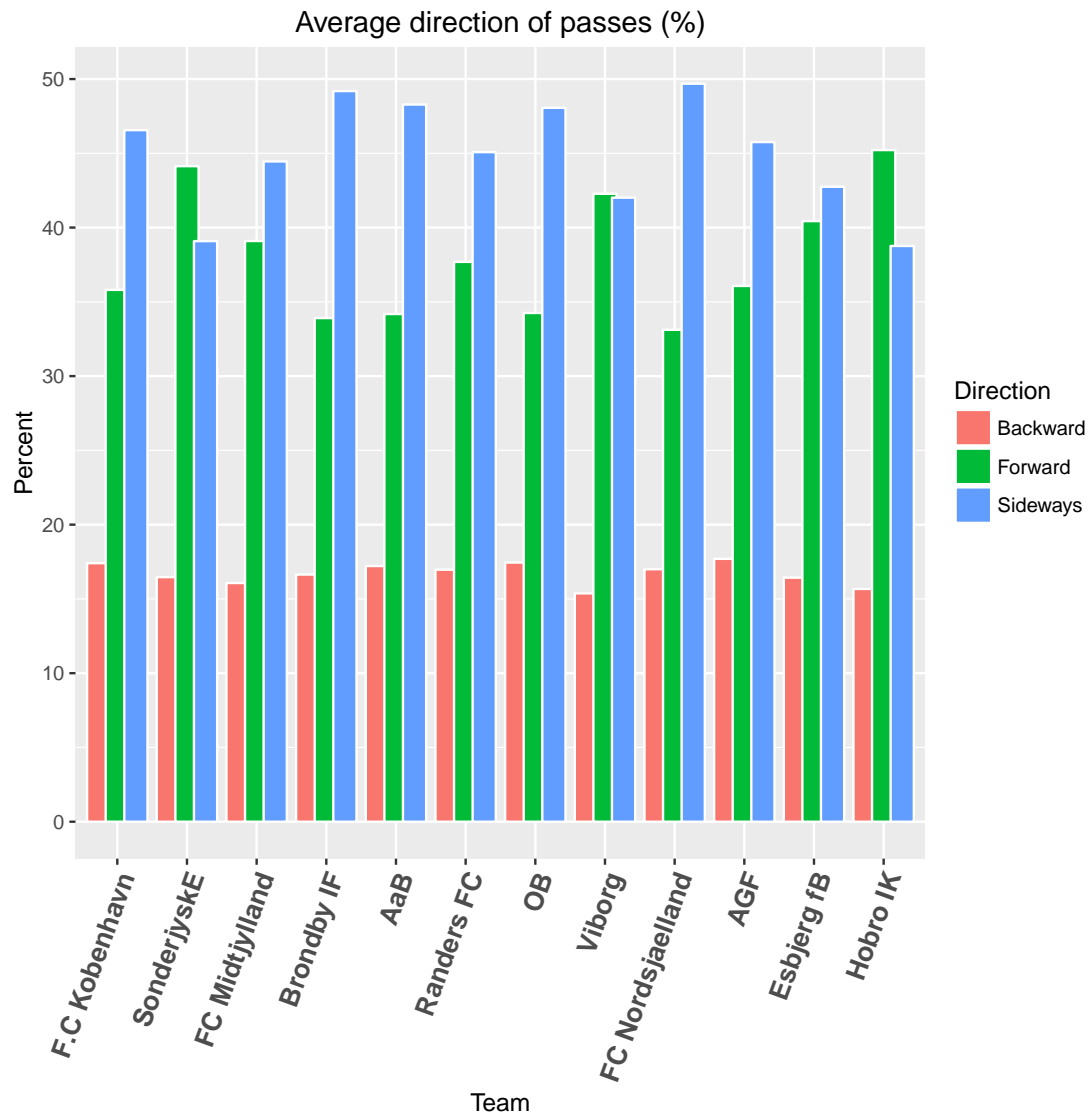


Figure 12: Grouped bar chart showing average pass directions per team

6.3.2 Code

The bar chart, as well as the data processing for the chart, was created in R. This was done by first selecting the variables in question from the Excel sheet, then splitting it out into 12 tables (one per team). The variables were then summed, and each summed variable divided by total number of passes, to get the percentages of short, medium and long passes. This was again done by creating variable and value lists, adding elements to these lists in a loop, and then creating a data frame. ggplot2 was then used to create the bar chart, in much the same way as the other grouped bar charts.

6.4 Passes -Length

6.4.1 What-why-how

A grouped bar chart that shows the length of passes of a team, on average, divided into three categories - less than 17 meters, between 17 and 34 meters, and passes greater than

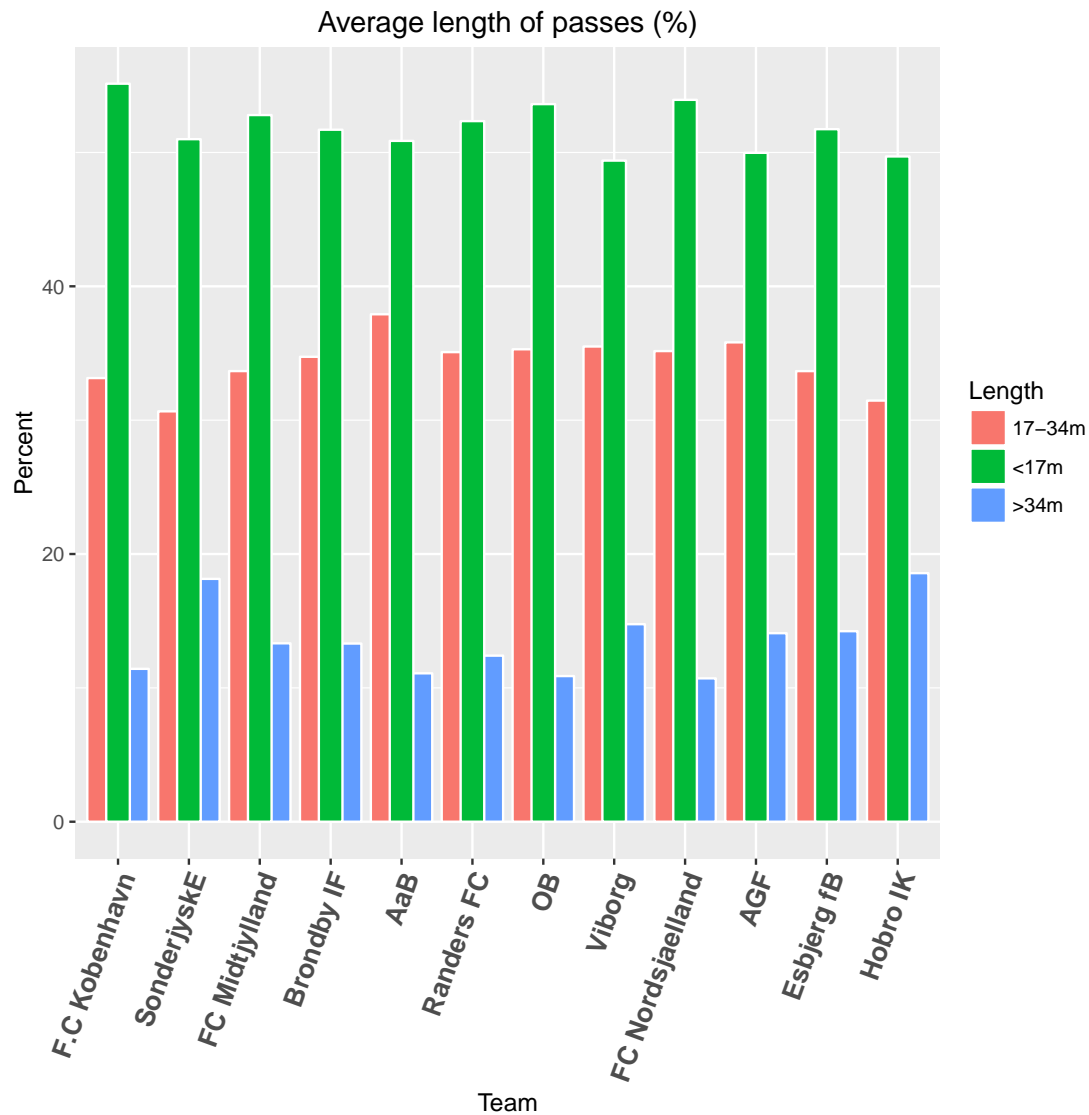


Figure 13: Grouped bar chart showing average pass length per team

34 meters. These three variables are grouped by teams, and the groups are sorted, based on the team's ranking in the league. The purpose of this idiom is to see if there is a correlation between the length of a team's passes, and their rank in the league. Summing the average passing lengths of a team results in 100

6.4.2 Code

This chart was made completely in R, by first selecting the desired variables from the Excel sheet, then splitting this data into 12 sets, one per team. The columns were then summed, then divided with the total number of passes, to get the different percentages. This was done in a loop, by adding variables and values to predefined lists, instead of iteratively adding data to a data frame, which is not efficient. By creating a data frame of the lists after iterating, a new data frame is not created after every iteration, thus increasing efficiency. The order of the groups is again decided on the levels of the data frame, and thus these are reordered, based on the ranking of the team in question.

6.5 Shots -Head vs Foot

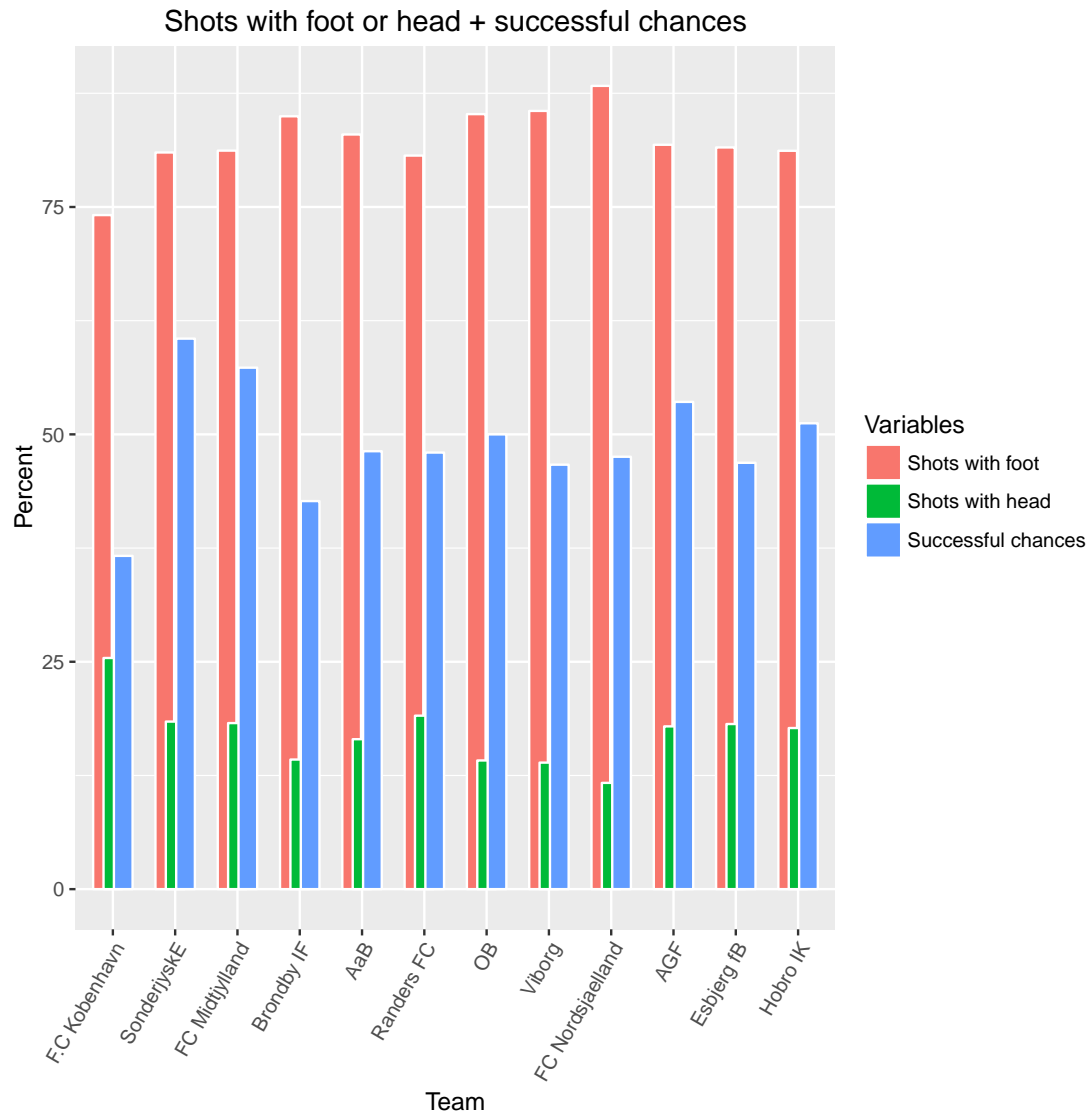


Figure 14: Grouped bar chart showing the ratio of headers and shots with foot as a percentage of total shots, together with the percentage of chances that results in goal

6.5.1 What-why-how

This idiom shows how many of a team's shots are taken with the players' feet, versus their head, as well as the ratio of goals per chance ('successful chances'). The groups (teams) are ordered based on their ranking in the league, with the highest ranking team as the first group. This visualization was created to see if there is a correlation between a team's ranking, and how many shots are headers, as well as how many of their chances are successful.

6.5.2 Code

As with the other grouped bar charts, R was used exclusively, with the Excel sheet as input data. The variables in question were selected, then split into different tables for each team. Then the variables were summed, and shots from head and foot were divided by total shots. The successful chances were calculated by dividing the total number of goals with the total number of chances. ggplot2 was utilized to create the grouped bar chart, ordering the groups by the ranking of the team, with the highest ranked team as the first group.

6.6 Shots -Inside box vs Outside box

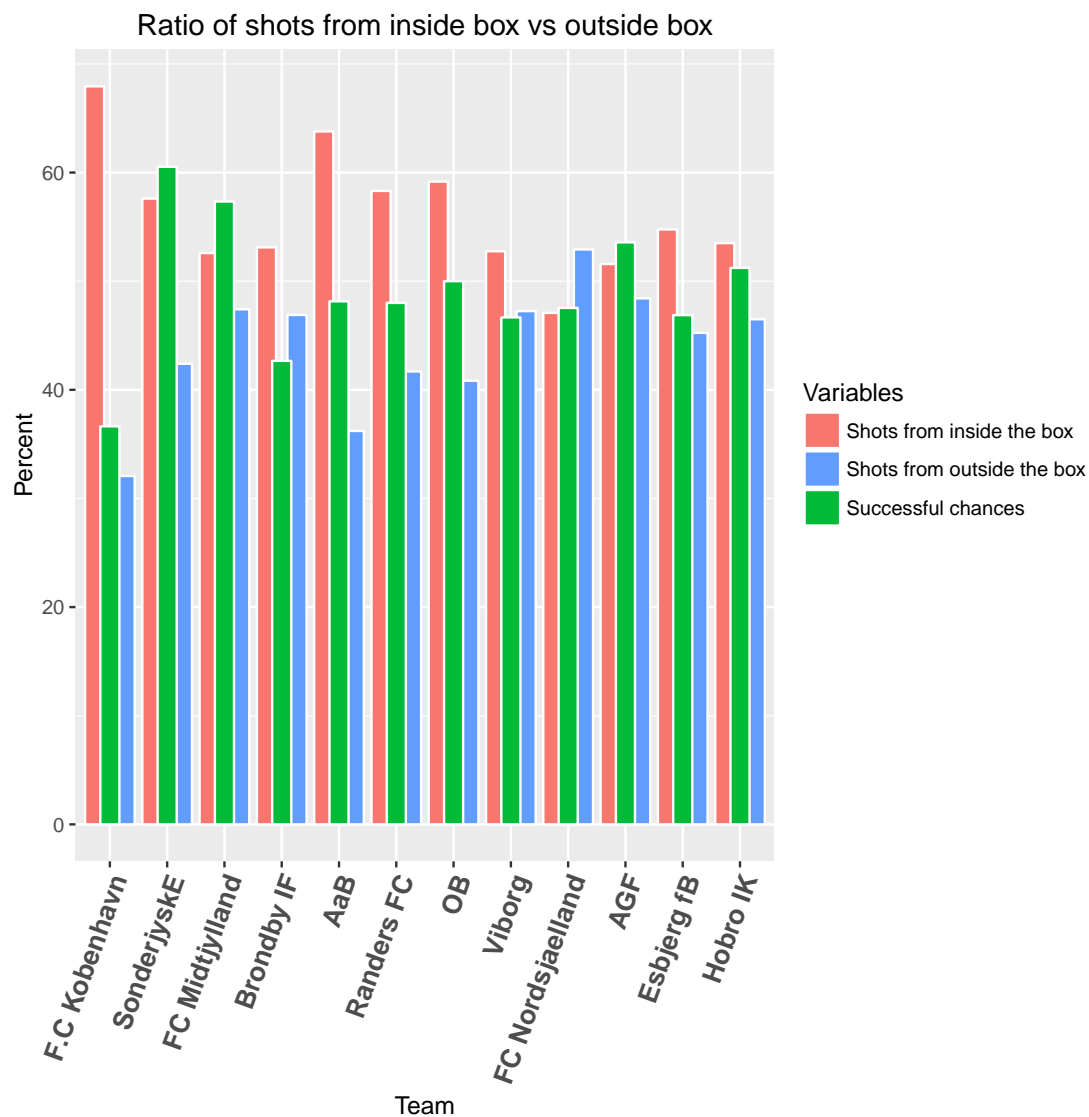


Figure 15: Grouped bar chart showing how many shots were taken from inside the box and outside the box, as percentages of total shots, as well as the percentage of chances that result in goal.

6.6.1 What-why-how

This visualization shows how many of a team's shots are taken from inside the box and outside the box, and the percentage of chances that are successful. It was created to explore possible correlations between a team's ranking, and where the shots originate from. The bar charts are grouped, based on a team's ranking, with the highest ranking team as the first group (F.C. København at the time of writing).

6.6.2 Code

The bar chart was created in almost the same way as the other grouped bar charts, utilizing R for both data processing, and creating the graph (with ggplot2.) Data was selected from the Excel sheet, and then split into the different teams, before summing the variables. The shots from inside/outside the box were then divided by total number of shots to get the percentages. Successful chances were created by dividing total number of chances with total number of goals. As with the other bar charts, the data was processed in a loop, processing one team at a time, appending the values to a list, and then creating a data frame outside the loop, with data from all the teams. ggplot2 was then used on this data frame to create the chart.

6.7 Success Rates

Comparison of Success Rates

Choose Two Teams to Compare

After selecting two teams to compare, you can pick a point in the radar chart to compare to all other teams

Estbjerg IB | SønderjyskE

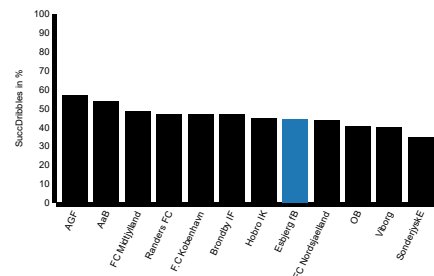
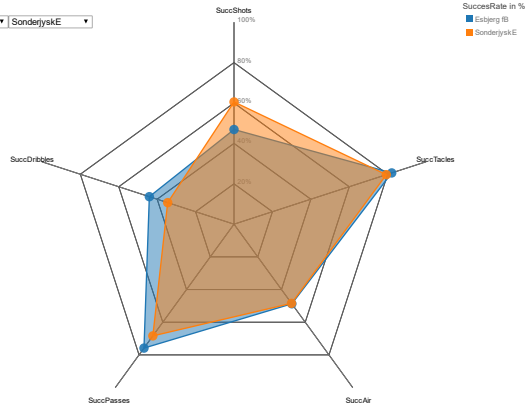


Figure 16: Idiom showing the comparison of the success rates using a radar chart for comparing two teams and bar chart for comparing with all other teams in the league

6.7.1 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 16. Concerning magnitude channels we position the marks on a common 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.

6.7.2 Code

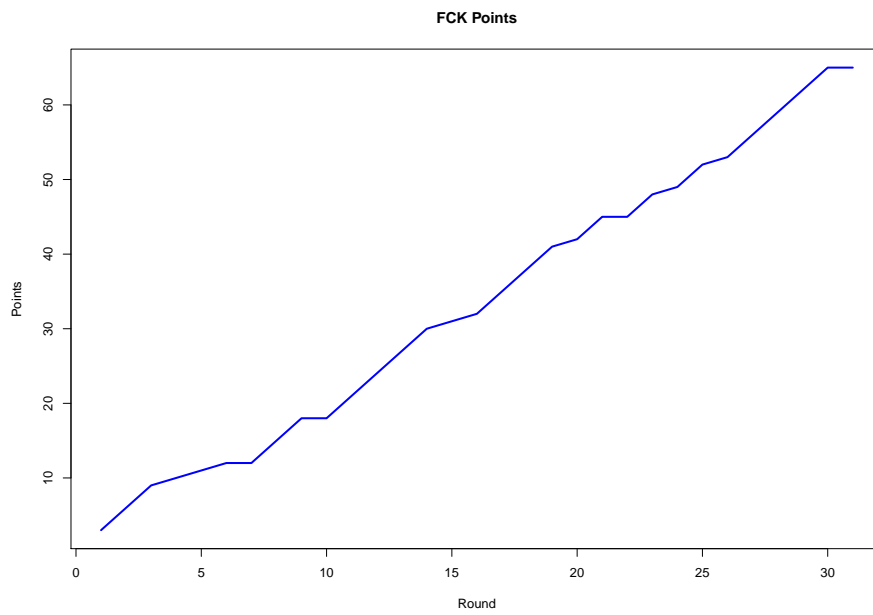
The R code is quite simple. In broad strokes it does the following: Read the data file, convert columns to be numeric, sum the data by team (before player wise), add and calculate the success columns, remove the unnecessary columns, restructure the data to a format that fits the radar chart and write 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.

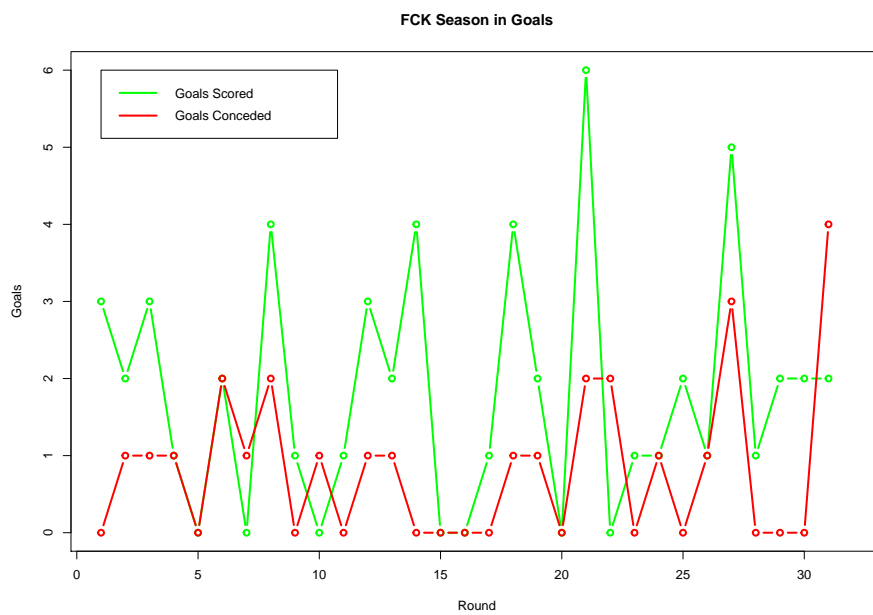
6.8 Season Review FCK

6.8.1 What-why-how

This visualization is a combination of two line graphs. It visualizes the amount points the Danish football team F.C. Copenhagen has accumulated throughout the season and the amount of goals scored and conceded throughout the season. Why this visualization? For this particular visualization the goal is to illustrate a team's overall evolution throughout a season. Each of these visualizations take a table as an input. The reason these tools are being used is to discover a trend or certain outliers. For the points graph, the data is ordered. It shows the progression of the team's point total throughout the season. For the goals graph, the two different lines are separated to illustrate the success of the team offense and defence. The colours in this graph are chosen based of human perception, as green is perceived as a positive colour, hence representing the positive event of scoring a goal, and red is perceived as a negative colour. The raw data is also reduced, as the visualization only shows the data from F.C. Copenhagen's season.



(a) some text



(b) Some text

Figure 17: some text

6.8.2 Code

These charts are created solely using R. First, the data file is loaded into R. Second, we use the functions in R to create a plot, and to create a line in the plot.

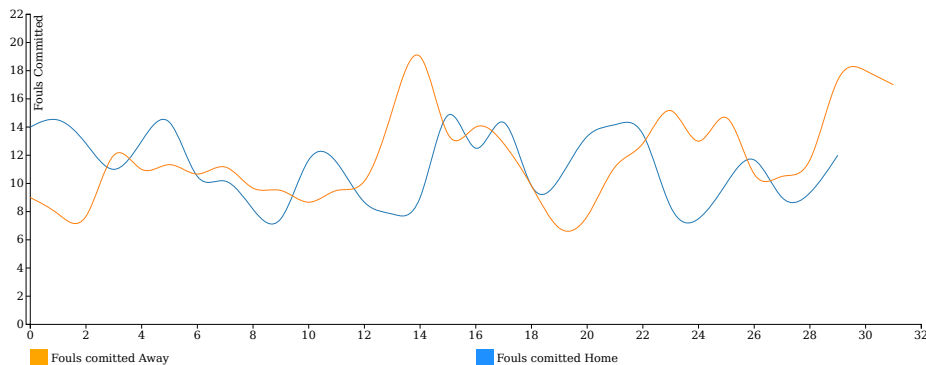


Figure 18: Some text

6.9 Fouls Committed over season

6.9.1 What-why-how

In hopes of discovering a trend between how far in a season a team is, and how many fouls they commit, we plotted these two variables together. A choice was made to separate matches where the teams are home, and the matches where they are not. This was done because there could be a difference in confidence and play-style, as well as there not being the same amount of matches played home and away. As seen on the chart, there is a lot of rising and falling. In order to conclude anything, alot more data would be needed than the few seasons we have access to at this point. However, the rising and falling could indicate difference in aggressiveness based on what team is their opponent in the match, we haven't quite gone deep enough go give any definitive answers as to what teams they in general prefer to be aggressive against.

6.9.2 Code

The code behind this chart lies in gathering and processing statistics for a team over multiple rounds, then filtering it for everything else than the foul statistics, while still preserving the right order of the matches. After that is done, it is put in D3, with some relatively simple code for a line chart.

6.10 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

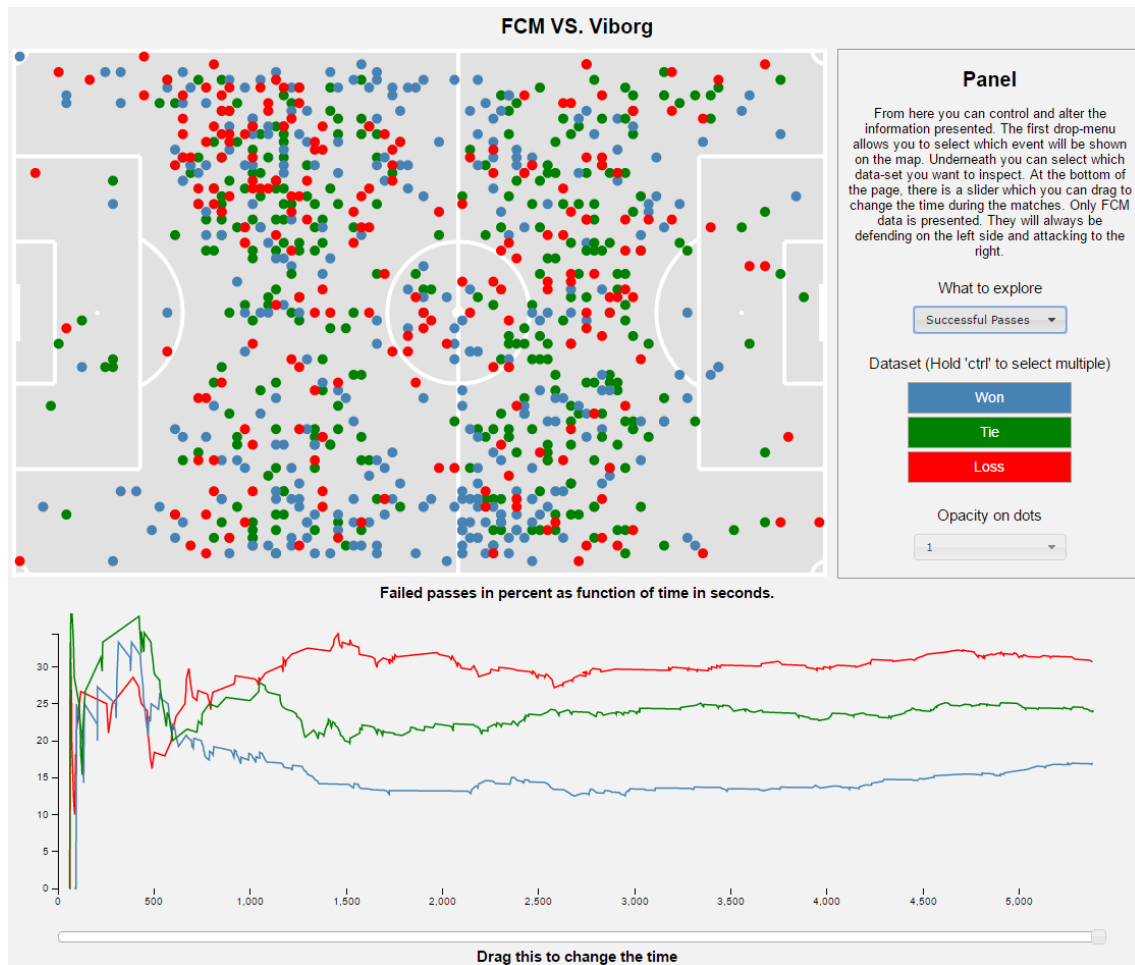


Figure 19: Kewl

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 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

6.11 Nationality of players

Nationality of the players

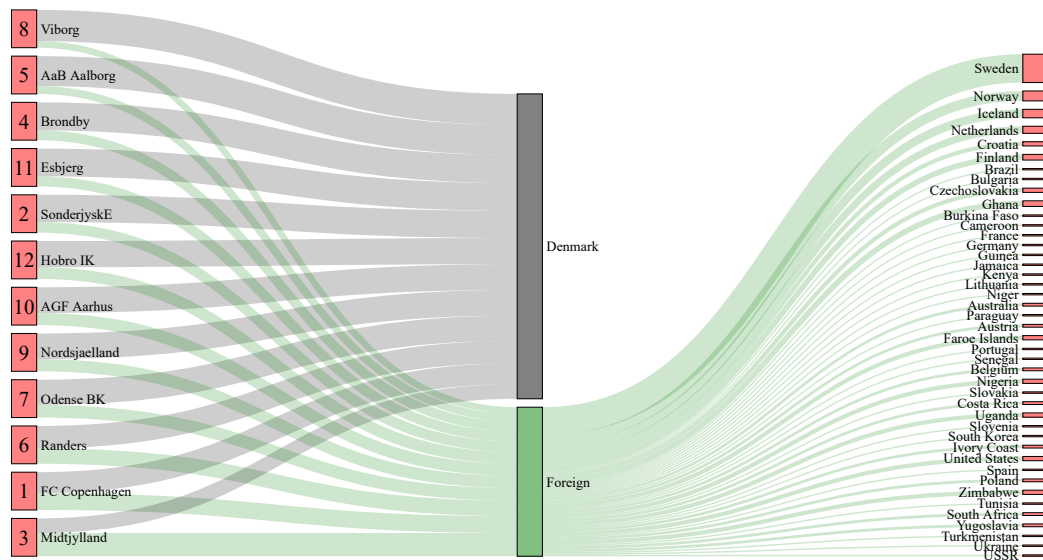


Figure 20: Sankey-Chart showing the distribution of Danish vs. foreign players on the different teams

6.11.1 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 20.

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.

6.11.2 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.

6.12 Success Rates Animated

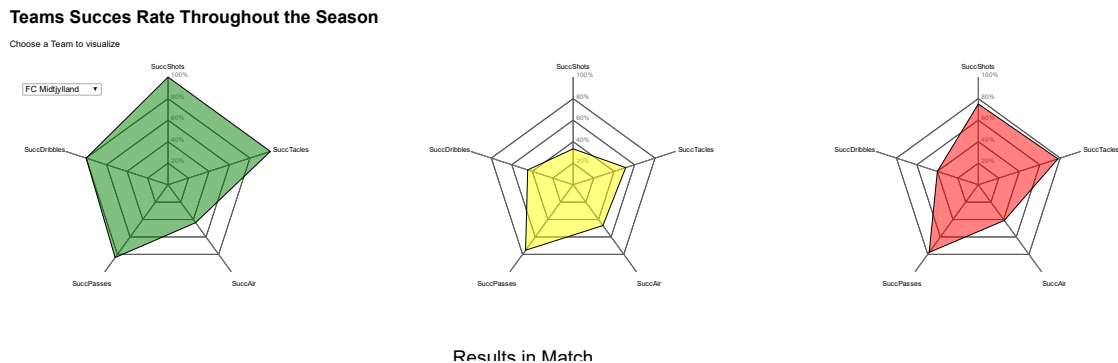


Figure 21: Comparison of a Teams success rates throughout the season, split in wins, draws and losses

6.12.1 What-why-how

Figure 21 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.

6.12.2 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.

6.13 Custom Bubble Chart

6.13.1 What-why-how

What. The data being presented in this visualization (figure 22) is the statistics of the teams from the current season. This 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 each other. 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 and the traditional bubble chart is that this visualization allows the user to interact with the visualization and change the parameters. The time complexity of this specific implementation is $O(n \lg n)$ where n is the amount of different parameters available to choose from. 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 the circles 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 sorted based on the third parameter, and the x coordinate is assigned. Since we sort 3 times, the complexity is going to be $O(n \lg n)$, assuming that we can sort in $O(n \lg n)$ time. While there might be smarter implementations than this very straightforward approach, it is not of any

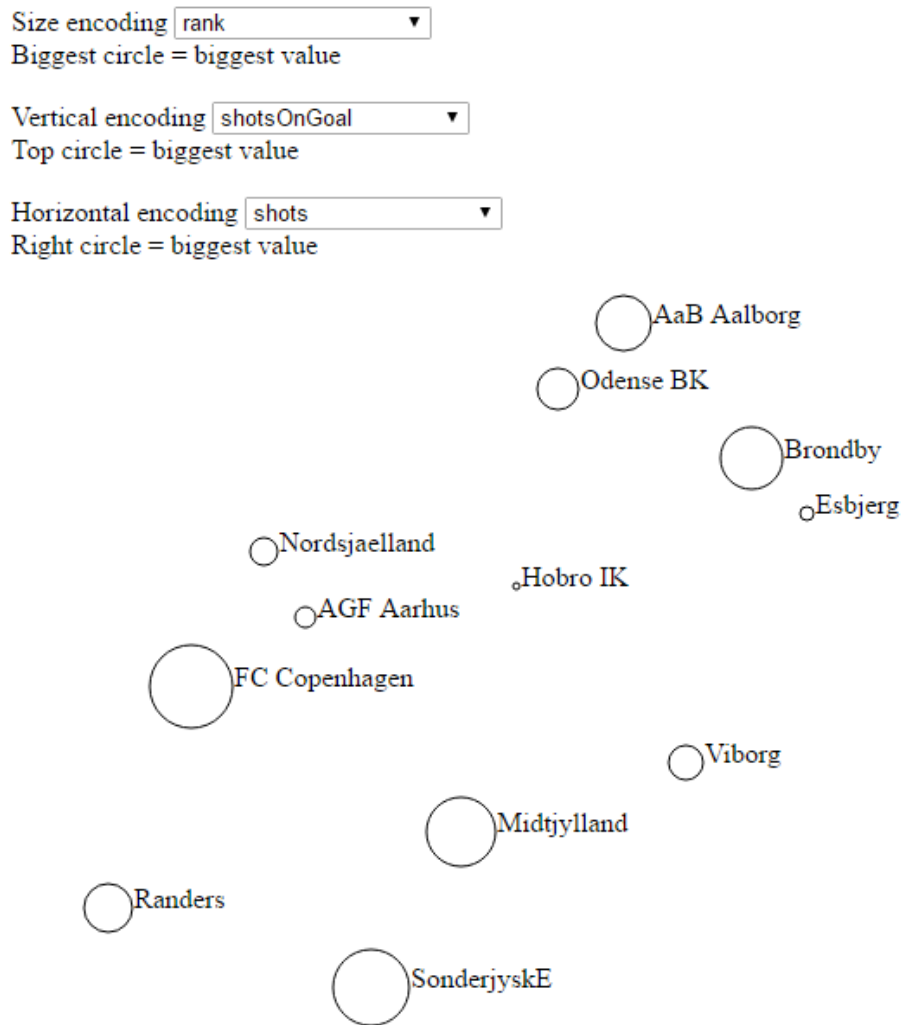


Figure 22: “Custom bubble chart”. Visualization of statistics of teams from the current season. Encodes rank, shots and shots on goal

concern at the moment, as the different number of parameters a user can choose from in football data is very limited. The smart thing about this 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 visualization is based on. 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 preprocessed in R and exported in the format that was described above.

6.13.2 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 circles 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).

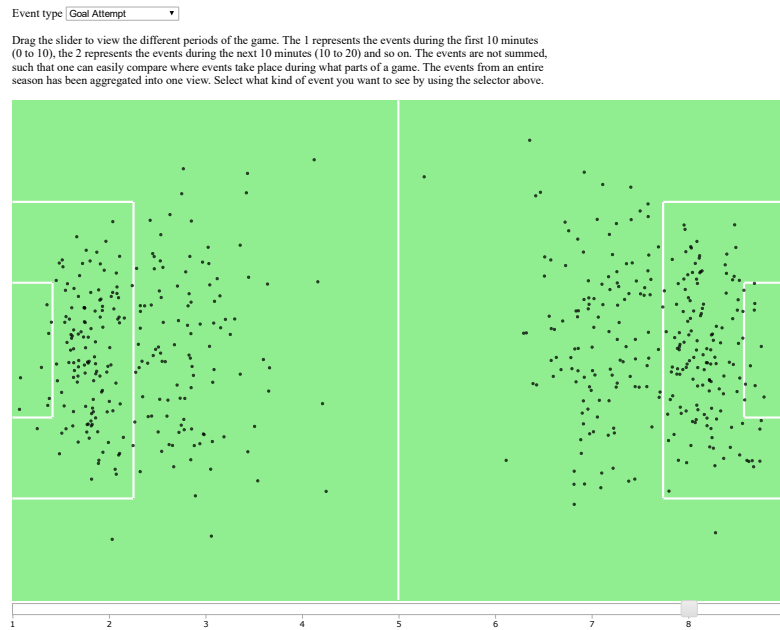
```
1 updateDimensions(objects, sizePar, xPar, yPar, distance) {
2   sortByParameter(objects, sizePar)
3
4   currentRadius = 1
5
6   for(object in objects)
7     object.radius = currentRadius
8     currentRadius++
9
10  sortByParameterReverse(objects, yPar)
11
12  prevRadius = 0
13  prevY = 0
14
15  for(object in objects)
16    object.y = prevY + prevRadius + distance + object.radius
17    prevY = object.y
18    prevRadius = object.radius
19
20  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 }
```

Figure 23: Pseudocode for updating the circles' parameters

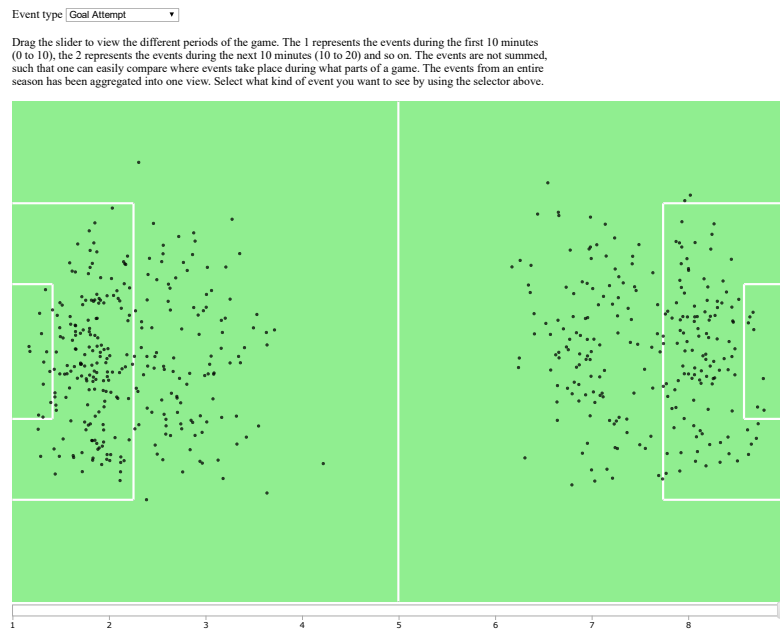
6.14 Field Events

6.14.1 What-why-how

What. The data being presented in this visualization (figure 24) is event data from a football season split into intervals. 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



(a) Goal attempts during the 70 to 80 minutes.



(b) Goal attempts during the 80 to 90 minutes.

Figure 24: Visualization of event data in intervals during a match. Goal attempts during the 70 to 80 minutes compared to goal attempts during the 80 to 90 minutes.

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 often occurs. 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 change throughout a match. The result of the hypothesis test is described later in the report in the discussion section.

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 directions. 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 coordinates. To prevent this from disturbing the meaning of the data, two things have been done. The first thing is that the opacity of each point is lowered a little bit, meaning that points are actually a bit transparent. This means that if two points are on top of eachother, the resulting 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 "jigging" and is often used when overlapping data are to be visually 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, a naive 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 bound 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 and whether it is 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 $O(n)$.

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 below. 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 matches 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 are then each split into 9 different data frames depending on the minute attribute of each event. The events during minute 0 to 10 are split into one data frame, the events during 10 to 20 are split into another and so on. These data frames are then saved to separate 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 processes 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.

6.14.2 Code

The visualization is made using the D3 javascript 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 follows. Where eventID is the id of a match and touch means an event within the game, for example a good pass.

7 Discussion

Our visualization allowed us to find certain patterns in the data and partly prove or disprove some common conceptions. One of the things we intended to look into was how the play-style of a team changed through out a match and to this end we found some tendencies which seemed to apply. The first one was the fact that late in a match, it seemed as a general tendency that teams had more goal attempts, we actually found a direct correlation between time passed and how many goal attempts were taken which may imply that the players either becomes more desperate and shoots more, or the defense-line simply perform worse as the game progresses. Also, it seemed that teams would actually make more failed passes at the very beginning of a match and at the start of the second half, which may be due to the fact they have to get "warmed up" and thus makes some mistakes at the beginning of each half. These theories are based on the results of summing up all touch-related data throughout an entire season and thus there is a large amount of data to back it up the notion. What could strengthen the hypothesis would be to look at other international leagues and maybe even multiple seasons to hopefully find the same tendency. If done successfully, it would improve the accuracy of the assertion. Another thing which was found, though more based on single instances, was that a winning team would play more horizontally along the sides of a football field, whilst when losing they would play vertically and close to the middle instead. This was deducted by analyzing three matches between FCM and Viborg which indicated that when a top-tier team as FCM played against a supposedly on paper worse team like Viborg, they would in their supposed state


```

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 +
23        1)*10)
24      toWrite = toJSON(currentTouches)
25      write(toWrite, touchDataFrame.name + i + ".json")
26 }

```

Figure 25: Processing event data in R for the Field events visualization

of winning play more along the sides whilst when losing would put more focus in the center. Also, the rate at which FCM failed their passes seemed to be rather high in respect to when winning or playing draw. By cross checking with a lost match against another top-tier team, namely FCK we found that this rate actually dropped dramatically, even though the outcome of the matches were much alike. Thus, it seems that when a team, or at least FCM, are losing to a much worse team, they make more mistakes in their passing. The cause can indeed be discussed, some suggestion could be that they are stressing more to get ahead as they are supposed to win the match. However, it may as well simply be due to the fact that they are playing way under their normal level on the given day in the first place. In regards to finding key differences between top-tier and low-tier teams, it turned out rather complicated to find any consistently different attributes. Some of the factors which we researched were based on summing some general player stats for all teams and compare them. Possibly counterintuitive, we were not able to find any direct correlation between how the best teams played nor the worse in opposition to each other, which lead to the notion that their may not be a single player attribute which leads to a winning team but rather a combination which is far more complicated. Another thing which were investigated was the distribution of players in terms of their nationality. We were not able to find any correlation between how many foreign players were on a team and their ranking but we saw small tendencies of teams having more foreigners depending on their establishment, meaning that teams which have been in the league for a while and thus have larger budget, tended to have more foreigners which may simply be due to the fact that they have the money and thus the option to purchase players from other countries.

When it comes to a team's change throughout a season, we were among other things hoping to find some correlation between how far in the season a game was, and how many fouls a team committed. This was with the mindset that a team would get more aggressive as the end of the season came. However, how far into the season a match were, didn't seem to affect how many fouls were committed, instead, it kept jumping up and down throughout the season. This could indicate that how aggressive a team play is more dependent on their opponent, than the position and/or time in the league. There were however a small spike towards the last couple of matches, but nothing bigger than some of the spikes from mid-season.

Despite using PCA and manually testing different variables that would make sense to change throughout a season, there simply weren't enough season-wide data to draw any substantial conclusions regarding how a team evolve throughout a season. In order to try and establish which variables determine the differences between winning teams and losing teams, a series of grouped bar charts were created, see figures 11-15. Figure 11 shows different types of attacks that lead to a goal, in percentages, grouped in teams by ranking order. For instance, Viborg scores as often from set plays as on quick attacks. On this chart, it is hard to notice any obvious tendencies between the higher ranked teams to the lowest, but there are a couple of interesting things to point out. FC Midtjylland, which is one of the top-tier teams in the league, is the team that scores most goals on set plays, compared to total goals scored, closely followed by AGF, which is currently the third-worst performing team this season. It seems as such, that instead of finding a grouping of these variables, we see them spread out across the teams, disregarding the ranking somewhat. FC Midtjylland scores more on quick attacks than AGF, so one is left to wonder if it is the combination of set plays and quick attacks that is more effective, or if other variables come into play, which is probably the most likely.

It does seem like there is a slight tendency towards better teams scoring less from possession based attacks, although there are some strong outliers, namely AaB, OB and FC Nordsjaelland, who all score more than 60 % of their goals from possession based attacks, but these outliers all lie somewhat in the middle of the league, with FC Nordsjaelland scoring much less on set plays compared to the other two teams.

Goals from open play, not attacking, are somewhat rare, as it is not often a team scores suddenly, without being the attackers. The same goes for penalties, although they are less rare than the aforementioned. For penalties, there does seem to be a slight tendency for lower ranked teams to score more on these, and this does make sense, as one could expect a lower ranked team to struggle with scoring enough goals, and so penalties account for a larger share of their goals. There are of course outliers, again FC Nordsjaelland, who didn't receive any goals from penalties, but instead a few goals from non-attacking open play.

Figure 12 shows which direction the 12 different teams prefer to pass in, divided between backward, sideways and forward passes. For the 2nd to the 4th team, it seems like there is a local negative linear correlation between forward and sideways passes, with forward passes being less for the 4th team, who pass sideways much more frequently compared to team 2 and 3. The same is seen for the last 4 teams, but in reverse order - teams pass less sideways and somewhat less backwards, but more frequently forward.

Figure 13 shows how far the teams pass on average, divided into three categories - $<17\text{m}$, $17\text{-}34\text{m}$, and $>34\text{m}$. On this chart, it is clear that FC Kobenhavn less frequently passes the ball long than the other top-tier teams, in fact there are only 3 other teams in the league that pass less long balls than FC Kobenhavn, and none of these are top-tier teams. It is interesting then, to see that SonderjyskE is one of the teams that pass long balls most

frequently, being the team in 2nd place. When you take into account that SonderjyskE was also among the teams passing the ball forward most frequently, this makes good sense - it is better to pass short balls backwards and long balls forward - a long ball backward would cost the team momentum. Hobro is the other team that looks like SonderjyskE in respect to both length and direction of passes.

Figure 14 shows how many shots are taken with the foot versus head, as percentages out of 100, as well as successful chances, which is calculated as the total number of goals divided by the total number of chances, and selected to indicate how successful a team is when trying to score. It clearly shows, counterintuitively, that the top ranking team, FC Kobenhavn, actually has the worst goal to chance ratio, which indicates that they are maximizing the number of chances, and not necessarily the quality of ditto. This team is also the most heading team in the league, leading by a relatively large margin.

SonderjyskE and FC Midtjylland stand out in terms of successful chances, indicating they take the opposite approach to quantity vs quality, compared to FC Kobenhavn. Apart from these two teams, it looks like there is a slight correlation between lower ranked teams having a higher ratio of successful chances, with a bit of variance. This is well within reason, since lower ranked teams create less chances, so they want to create the best possible chances in general. It also seems as though lower ranked teams shoot more with their feet than with their heads, but this is less clear, since the lowest 3 teams head about as much as the 2nd and 3rd teams.

Figure 15 shows where shots are taken from, divided into shots from inside the box, and shots from outside the box. Successful chances is based on the same data as in Figure 14, and calculated the same way.

From Figure 15 it is clear that the highest ranking team is also the team that shoots from inside the box most frequently. Based on the earlier charts, one can speculate that FC Kobenhavn concentrates on passing the ball forward with short passes, getting it inside the box, and then getting the ball into the goal, instead of trying to create chances by lurking outside the box.

As with the other charts, it looks - apart from SonderjyskE and FC Midtjylland - as if there is somewhat of a linear correlation between different variables and the ranking of teams. Here, it seems as if lower ranked teams tend to shoot more often from outside the box, but again the correlation is too weak to conclude anything meaningful.

We also investigated how a team has evolved throughout a season. Here, the biggest takeaway comes when looking at the amount of points FCK has after each round. This visualization helped illustrate the highs and lows of the team's season, and it showed how the team got off to a relatively slow start, only to get going and never looking back. We assumed that by looking at goal scoring and conceding in the first 10 games and compare them to the remaining games, would help illustrate the evolution in the team. However, the results are simply far too inconsistent to reach any conclusions of.

8 Conclusion

9 Usability

The visualization techniques and idioms we have presented in this report can be applied in many different settings. Some of the visualizations are very specific to football, for example the ones with a football field, but these can easily be adapted to visualize data from other branches of sports. The more generic ones we have made such as the variation of a bubble chart and the radar plots can easily be used to present other kinds of data which is not

necessarily from sport. This could be financial data, biological data etc. The visualizations we have made can specifically be used by a team to analyze how other teams play and adopt their strategy when they meet those teams, or to analyze in general how teams play throughout a match, and then see if they can learn something which can improve their own strategy. The visualizations can also be used on websites where normal football fans can view them and see if they can produce any interesting hypotheses, which they can try to test the next time they watch a game in the tv. The visualizations can be used by betting companies such that they can give their users tools such that the users can find out how they want to place their bets. The visualizations can also be used during live transmissions of football games in the tv, for example by showing statistics about the match during the intermission. In the academic setting, some of the visualizations can be used by psychologists for example, to learn something about the mentality of players during a game, for example by using the field representation of shots, to see where a player is likely to shoot from under certain conditions.

10 Bibliography

- [Bostock(2016)] Mike Bostock. D3: Data-driven documents, 2016. URL <https://github.com/d3/d3>.
- [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>.
- [Iliinsky and Steele(2011)] Noah Iliinsky and Julie Steele. *Designing Data Visualizations*. O'Reilly Media, 2011. ISBN 9781449312282;.
- [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.
- [Simmering(2013)] Jacob Simmering. How slow is r really?, 2013. URL <http://www.r-bloggers.com/how-slow-is-r-really/>.
- [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.

Process Evaluation

At our first meeting in the group we made the group contract. We discussed how we would like to distribute our work and what we should do to cooperate in the most successful way. Early in our process we also met with the company Prozone, which is the company providing us our data. Here we got information about their data, current tools, what they could be interested in and what they could offer us. It seemed very promising but unfortunately the process of getting the data was a long one. It took around a month to get access to the API, which of course was a challenge, because we did not know exactly what data we could get access to. We got some data from the beginning, but this was very limited. Because of that most of our visualisations are based on this test data, and scaled when we finally got access to more.

Because of the lack in data, we had focus on the theory the first weeks. We explored the theoretical materials by our own, and later on discussed them in the group. This has generally been the process of our work. Also when getting access to the API we could start to share knowledge about the data we could get. We have programmed our visualisations individually and helped each other to understand the programming languages. If one had problems we tried to solve these together. During the design process we also gave each other feedback on the specific visualisations.

At the midway seminar we had some ideas about what we would like to visualise, and had some base visualisations to show. There was some general structural feedback on our presentations, like missing slide numbers, slide titles and conclusion slide. We needed a greater theoretical knowledge too, so we would be able to answer any questions, and be able to explain every topic we mentioned. Lastly our supervisor emphasised the importance of using original graphics, which not all members did.

At one of our meetings we talked about which theoretical topics we would like to include in our report. Afterwards we delegate the different sections between us. In the end we read each others sections to check for spelling mistakes, typos but more importantly to agree on the theoretical content. Each member have written their own results section, and contributed to the discussion and conclusion section from this. We agreed on when to be finished with the theory sections, but this deadline was not respected by most of the group members. This meant that the last week has been somewhat stressful for some group members.

Generally the group have met a couple of times a week, to discuss progress and theory. We evaluated how long we had come, and what the next step was. We met once a week with our supervisor to get feedback on our work. The group meetings have typically been placed before and after each meeting with our supervisor, to respectively discuss what our agenda to the supervisor meeting should be, and what we should do next based on the feedback from our supervisor. At almost every group meeting every member was present, and most of the time arriving on time. We have generally given members who was late a few minutes before we starting the meeting.

We have had no major conflicts, only small disagreements with our supervisor about what we thought the project should be about, and which topics we should include. Our supervisor wanted us to include some machine learning, which we thought was a bit off topic. In the end we decided not to include it. The group have worked together in mutual respect for each other, no conflicts have risen.

We had given a time line over the project but have had big problems following this, because of the late data arrival. A lot of time has also been wasted on email correspondence with Prozone concerning the data. This also means that the process has been somewhat

even more compressed and pushed to the end.

Concerning the mandatory L^AT_EX and library courses we have the following comments. The library course was informative, and gave us some knowledge of where to find materials and what to use. We had already plenty of materials, so we have not used the knowledge much. The L^AT_EX courses was not so informative, because we already where used to writing in L^AT_EX, only few informations where valuable for us. The course about poster creation in L^AT_EX was not useful at all. A few examples was explained, and the we could begin to make our posters, and the instructor could help us. The structure would be great if the course had been later in the process, we did not begin creation of the poster because we was not ready for it.

When having to evaluate the process it is our opinion that we should have been working even more together in the same room discussing our temporary results and discoveries. In the last couple of weeks we worked more intensively together, which gave great results, and which showed new opportunities which we could have followed if we had hat the time. We think it worked to study the theory individually and then talk about it, if we had any doubts, but once again it could have been a great idea, maybe to study different parts of theory and then explain it to the others. In this way we could have covered a larger field without having to individually read it all by our self. This would also be a great way to practise explaining the theory for others. In relation to the deadlines not being respected, we should have been more honest with each other, and talked about the why this was the case. We should earlier in the project have made deadlines for bringing concrete visualisations, so we was more on the same level in the process and could have improved our visualisations even further. If we had done these things we might even had enough time to expand the project into machine learning.