Development of a Football Data Visualisation tool using R and Shiny

Tim Beary

N00192726

Supervisor: Cyril Connolly

Second Reader: Catherine Noonan

Year 4 2022-23

DL836 BSc (Hons) in Creative Computing

Abstract

The aim of this project was to develop a football players data visualisation tool. As a passionate fan of sport, I've always been interested in how data can be used in sport, for example how visualisations can be used to explain player performance. The purpose of this application is to provide a data visualisation tool that allows users to view and interact with various plot or charts about football players. The application was developed using R and the web application framework for R called Shiny. R allows for the creation of various plots/graphs and charts that can visualise data and Shiny provides a framework to build web application to allow users to interact with these plots/graphs and charts. The application scrapes data and then processes this data to be used in the application. It then displays this data in a series of scatter plots and pizza charts. Having developed the application, user testing was then carried out which was overall very positive, with a few minor issues that could be addressed.

Acknowledgements

I would like to use this section as an opportunity to thank my supervisor Cyril Connolly for his valuable help and feedback during the development of this project. The experience and knowledge Cyril has of this area was incredibly helpful during this project. I would also like to thank the other lecturers who have provided support by running various workshops throughout the development process. Finally, I thank those who took part in the requirements survey and those who were involved in the user testing whose feedback was very appreciated.

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Course Director.

WARNING: Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not to leave copies of your own files on a hard disk where they can be accessed by other. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline.

Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties.

The following is an extract from the B.Sc. in Creative Computing (Hons) course handbook. Please read carefully and sign the declaration below

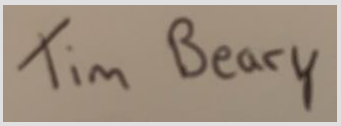
Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.

DECLARATION:

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student : Tim Beary

Signed



Failure to complete and submit this form may lead to an investigation into your work.

Table of Contents

Table of Figures

1 Introduction

The aim of this project was to develop a football data visualisation tool using R and Shiny. R is a statistical programming language that is used for data analysis and visualisations, Shiny is a web application framework for R that can be used to develop interactive web apps with R. The use of data in sport has become increasingly more popular in recent years. Not just among sports organisations, but also among fans and analysts. Data is now being used to help judge player performance.

This report will discuss the entire development process of this project. The report will begin with a research chapter which looks at various data visualisation techniques, and then discusses the use of data visualisation in sports. The next chapter will discuss the requirements for the application and how these requirements were gathered. Chapter 4 will outline the design for the project. This chapter will first look at the program design and then the user interface design. The following chapter is the implementation chapter. This will describe the development process and how the application has been developed. It will contain a number of code snippets which show how the various features and functionality have been coded. The next chapter will discuss the testing that was done on the application. This chapter is divided into two sections. The first will focus on the functional testing of the application which checks whether the functionality works as it is expected to. The next section will outline the user testing done on the application. This will involve giving the users the application to use and getting feedback from users. The user testing focusses on the UI and the user experience with the application. Finally, the project management chapter will outline how the project was managed during the development process. The Scrum methodology will be used during this project. The project will be broken down into sprints, with each sprint lasting two weeks.

The application will scrape data to be used, and then visualize this data using several interactive visualisation techniques with R and Shiny. The two main features of the application will be a scatter plot feature which displays player data based on user inputs and filtering. The other feature will be a player pizza chart which will visualize how a player is performing in a variety of different statistics.

2 Research - Examining data visualisation techniques and the use of data visualisation in sport.

2.1 Introduction

Few (2007) describes data visualisation as “a powerful means both to make sense of data and to then communicate what we’ve discovered to others”. Data visualisation can be a crucial tool for understanding and communicating large and complex data. It allows the data to be presented in a visual format, making it easier for individuals to understand and form opinions from the information. In the field of sport, data visualisation plays a significant role in the analysis and interpretation of data related to performance, team strategy, and more. This research chapter will discuss the topic of data visualisation and also view it through the prism of sport for the purpose of developing a data visualisation application using football data for my final year project. The first section of this chapter will involve the research of data visualisation techniques. There are a variety of techniques used in data visualisation, including bar charts, line graphs, scatter plots, and heat maps. Each data visualisation technique can serve a specific purpose and can be used in different ways to effectively communicate data. The next section will research and discuss the role data visualisation has in sports. The use of data visualisation in sport has increased significantly in recent years, with it becoming an important tool for many in sports. It allows for a more in-depth understanding of athletic performance, team strategy, and other factors that can impact the outcome of a game or competition. Data visualisation can also be used to identify trends and patterns in data, which can help decision-making and strategy development.

2.2 Data Visualisation Techniques

Data visualisation techniques are tools that allow users to represent data and information in graphical or visual form. These techniques can be used to identify patterns and trends and help communicate these patterns or trends during the decision-making process. There are many different types of data visualisation techniques, ranging from traditional methods such as bar charts and line graphs to more modern techniques like heat maps and network graphs.

2.2.1 Traditional Techniques

2.2.1.1 Pie Chart

Pie charts are a very popular technique and the one of the more basic data visualisation techniques used (17 Important Data Visualization Techniques | HBS Online, 2019). Pie charts are circular charts that are divided into several slices to represent different data categories. The size of each slice represents the quantity of the data being shown, making it easy to compare different categories. They are useful for visualising how different parts make up a whole. Due to Pie charts simplicity, they are best suited to be used for people who are unfamiliar with the information being displayed,

therefore pie charts are not suitable for complex data that requires in depth explanations (17 Important Data Visualization Techniques | HBS Online, 2019). Two variations of pie charts include Donut charts and Exploding pie charts (Gandhi & Pruthi 2020).

2.2.1.2 Line Chart

Line charts are visualisations of data that show the changes in a single statistic over a period of time. They consist of a series of connected data points, called markers, that are plotted on along the x-axis (horizontal) and y-axis (vertical). According to Gandhi & Pruthi (2020) line charts are most effective when attempting to show a change in a variable or variables. One of the main benefits of line charts is that they can show the overall trend of the data clearly and concisely.

2.2.1.3 Bar Chart

Bar charts are visualisations of data that compare different categories or groups. They consist of a series of bars, each of which represents a separate data category or group. The length of each bar is proportional to the quantity it represents, making it easy to compare the sizes of different categories. Gandhi & Pruthi (2020) say that bar charts are "not very effective when the amount of data is very huge". Another drawback of bar charts is that "labelling and clarity can become problematic when there are too many categories included" (17 Important Data Visualization Techniques | HBS Online, 2019).

2.2.1.4 Scatter Plot

Scatter plots show the relationship between two numerical variables. Scatter plots are useful for visualising the relationship between two variables, such as the relationship between price and quantity demanded or the relationship between weight and height. They can be used to identify patterns and correlations in the data, and to see how one variable changes as the other variable changes (17 Important Data Visualization Techniques | HBS Online, 2019). Scatter plots are best when used with large datasets, "since it's often easier to identify trends when there are more data points present" (17 Important Data Visualization Techniques | HBS Online, 2019).

2.2.1.5 Histograms

Histograms are visualisations of data that show the distribution of a single numerical variable. They can be used to identify patterns and trends in the data, and to see how the data is distributed across different ranges of values. Histograms are most effective in presenting the frequency of a particular value (17 Important Data Visualization Techniques | HBS Online, 2019). However, histograms are not suitable for visualising data that has a large number of data points, as they can become cluttered and difficult to read. They are also not suitable for comparing data between different categories or groups.

2.2.2 Modern Techniques

2.2.2.1 Heat Maps

Heat maps consist of a grid of cells, with the colour of each cell indicating the value of the data. Heat maps can use a range of colours, from cool colours like blue and green to warm colours like red and yellow, to represent the data, with warmer colours indicating higher values and cooler colours indicating lower values. The use of colours allows viewers to quickly and easily identify trends. This means a clear legend is required (17 Important Data Visualization Techniques | HBS Online, 2019). According to Gandhi & Pruthi (2020) heat maps can be considered a traditional data visualisation technique as well as one that can be used for modern data visualisations.

2.2.2.2 Interactive Visualisations

Dimara and Perin (2019) defines interactive visualisation as "Interaction for visualization is the interplay between a person and a data interface involving a data-related intent, at least one action from the person and an interface reaction that is perceived as such. "

Interactive visualisations allow users to manipulate the data or the visualization itself in some way. Interactive visualisations are useful for exploring data in a more flexible and dynamic way. They can allow users to drill down into specific data points or categories, or to compare different data sets or visualisations side by side (Qin et al., 2019). They can also provide a more engaging and immersive experience for the user, helping to make the data more meaningful and relevant.

Interactive Visualisations can be done using tools like Tableau, D3.js or R Shiny.

2.2.2.3 Network Visualisations

Network visualisations are data visualisations that show relationships between entities. They consist of a series of nodes, which represent the entities, and a series of links, which represent the relationships between the entities. Network diagrams can be used to represent many different types of data, including social networks, computer networks, and biological networks (17 Important Data Visualization Techniques | HBS Online, 2019). A Tree Map is an example of a network visualisation.

2.3 Role of Data Visualisation in Sport

According to Perin et al. (2018) the role of data visualisation in sport can be broken down into two different categories: 'Analytical' or 'narrative'.

Data visualisation plays an important analytical role in sport by allowing for the interpretation and analysis of large and complex data sets. By presenting data in a visual format, it can be more easily

understood and analysed by all stakeholders. In sport, data visualisation can be used to analyse a range of data, including performance data, tactical data, and statistical data. Performance data can include metrics such as passing accuracy, shooting percentage, and speed, and can be used to understand the strengths and weaknesses of individual players or teams. Tactical data can include data on formations, attacking and defensive strategies, and player movements, and can be used to evaluate the effectiveness of different tactics. Statistical data can include data on scoring, possession, and other game metrics, and can be used to analyse trends and patterns in a team's or player's performance. Perin et al. (2018) say that the analytical role of data visualisation in sport focuses on "understanding an individual's or a team's performance". It's about seeking 'Why' a team/individual is successful. They say that data visualisations can be used in scouting situations, helping to identify opponents' tendencies in order to develop a gameplay. Similarly, it can be used for self-scouting purposes. To learn about personal tendencies and behaviours to help benefit performance.

In addition to its analytical role, data visualisation can also play a narrative role in sport by allowing for the creation of interesting story lines using sports data. By presenting data in a visual format, it can be used to illustrate and explain complex ideas and theories in a way that is easy to understand and engaging to a wide audience. Perin et al. (2018) suggest that "Effective visualisations can communicate nuances of performance data that cannot be conveyed easily by statistics.". One way that data visualisation can be used to create narratives in sport is by highlighting the performance of individual players or teams. For example, a visualisation of a player's scoring record over time can be used to tell the story of their career, highlighting their achievements and milestones. Similarly, a visualisation of a team's performance over a season can be used to tell the story of their campaign, highlighting the factors that contributed to their success or failure. An example of this is showing league tables in football (Perin et al., 2018).

Alternatively, Du & Yuan (2020) break the roles of data visualisation in sport down into different categories. They categorise the roles as presentation, comparison and prediction roles.

The presentation role of data visualisation is the most commonly used role to display and present data. For example, a visualisation of a player's scoring record over time can be used to illustrate their performance and achievements. Similarly, a visualisation of a team's performance over a season can be used to highlight their successes and failures and provide context for their performance. Du & Yuan (2020) further sub-divide the presentation role down to: Presentation for trajectory information, Presentation for player's performance, Presentation for special events, Presentation for game information and Presentation for other information.

The comparison role of data visualisation in sport is about creating visualisations that compare the performance of different players or teams. For example, a visualisation of a player's scoring record over time can be used to compare their performance to that of other players, and identify trends and patterns in their performance. Similarly, a visualisation of a team's performance over a season can be used to compare their performance to that of other teams, and identify trends and patterns in their performance. Du & Yuan (2020) sub-divide the comparison role into Player's performance comparison, Behaviour events comparison and Game information comparison.

Finally, the prediction role Du & Yuan (2020) outline allows for the forecasting of future performance based on past data. By presenting data in a visual format, it can be more easily understood and analysed, and can help to identify trends and patterns that may inform predictions about future performance. The prediction role is further sub-divided into categories including Game result estimation, Sports event prediction and Game tactics decision-making.

Overall, the role of data visualisation in sport is significant. It allows for more effective communication and interpretation of data, and is an important tool for coaches, players, broadcasters, and fans alike. By presenting data in a visual format, it allows for a deeper understanding of the game and the performance of individual players, and can inform decision-making at all levels of the sport.

2.4 Conclusion

The purpose of this literary review was to research the topic of data visualisation and how it can be used in the world of sport with the aim of developing an interactive web application for visualising football data. This was done by first looking at the techniques used in data visualisation. Data visualisation is a valuable tool for representing and communicating data and information. There are many different types of data visualisation techniques, ranging from traditional methods like bar charts and line graphs to more modern techniques like heat maps and network graphs. Effective data visualisation is crucial for communicating findings and facilitating understanding, and the design of visualisations plays a key role in their effectiveness.

Next, research was carried out on how data visualisation plays a role in sports. By presenting data in a visual format, it allows for more effective communication and interpretation of the data, and can be used for a variety of purposes, including analysing performance, evaluating strategies, enhancing broadcasts, and analysing trends and patterns.

Overall, the literature suggests that data visualisation is an important and effective tool for the analysis and understanding of football data.

3 Requirements

3.1 Introduction

This requirements section will attempt to determine what the application should be able to achieve. Finding out what the users want the application to do will help achieve this. I'll start by examining some existing, comparable applications to see what we like and don't like about them. This will help determine which features to use and which ones to avoid for the application. I will also make a survey and use the responses to better understand what users want and don't want from the application.

Having this information, I will use it to establish both the functional and non-functional requirements for the application.

By interpreting what the user needs rather than what I as the developer want, the requirements section will aid in forming an idea of precisely what the application will perform.

3.2 Requirements gathering

3.2.1 Similar applications

3.2.1.1 FBREF

FBref is a football statistics website that provides advanced statistics on players, teams, leagues and competitions. The website was created by Sports Reference, which has similar statistics websites for other sports like basketball and American football. FBref are provided with all their data by the sports analytics company Opta. FBref is completely free to use and is the source of the data used in my application.

The main positive feature of the FBref website is its wealth of advanced data on players, teams and leagues. The website provides a large range of statistics like goals, assists, shots, passes and tackles, as well as advanced football stats like xG (Expected Goals), xA (Expected Assists) and progressive passes and carries. FBref offer these advanced stats on as many as 10 leagues and cups. The level of detail provided by FBref is the best available without paying for subscriptions. This data is regularly updated so users know they are getting the most up to date statistics.

Squad Standard Stats 2022-2023 Premier League Share & Export ▾ Glossary Toggle Per90 Stats

Squad Stats | Opponent Stats

				Playing Time				Performance								Expected				Progression		Per 90 Minutes									
Squad	# Pl	Age	Poss	MP	Starts	Min	90s	Gls	Ast	G+A	G-PK	PK	PKatt	CrdY	CrdR	xG	npxG	xAG	npxG+xAG	PrgC	PrgP	Gls	Ast	G+A	G-PK	G+A-PK	xG	xAG	xG+xAG	npxG	npxG+xAG
Arsenal	26	25.2	59.6	30	330	2,700	30.0	69	52	121	66	3	3	43	0	58.9	56.9	43.5	100.4	669	1651	2.30	1.73	4.03	2.20	3.93	1.96	1.45	3.42	1.90	3.35
Aston Villa	26	27.7	49.5	30	330	2,700	30.0	39	26	65	36	3	3	59	1	38.4	36.1	29.0	65.2	489	1003	1.30	0.87	2.17	1.20	2.07	1.28	0.97	2.25	1.20	2.17
Bournemouth	31	27.1	38.9	30	330	2,700	30.0	28	19	47	28	0	0	54	0	29.9	29.9	21.5	51.4	371	756	0.93	0.63	1.57	0.93	1.57	1.00	0.72	1.71	1.00	1.71
Brentford	25	26.9	43.4	30	330	2,700	30.0	46	27	73	39	7	8	42	1	44.0	37.8	29.7	67.5	295	888	1.53	0.90	2.43	1.30	2.20	1.47	0.99	2.46	1.26	2.25
Brighton	26	27.4	59.9	28	308	2,520	28.0	48	32	80	43	5	5	36	0	51.0	47.5	37.8	85.3	580	1388	1.71	1.14	2.86	1.54	2.68	1.82	1.35	3.17	1.70	3.05
Chelsea	32	27.2	59.1	30	330	2,700	30.0	29	20	49	26	3	3	67	3	39.0	36.7	30.0	66.7	624	1366	0.97	0.67	1.63	0.87	1.53	1.30	1.00	2.30	1.22	2.22
Crystal Palace	26	27.6	44.2	30	330	2,700	30.0	27	23	50	27	0	2	70	3	30.2	28.5	23.5	52.0	398	945	0.90	0.77	1.67	0.90	1.67	1.01	0.78	1.79	0.95	1.73
Everton	28	27.3	43.5	30	330	2,700	30.0	22	18	40	20	2	2	64	1	32.5	31.0	23.5	54.5	399	819	0.73	0.60	1.33	0.67	1.27	1.08	0.78	1.87	1.03	1.82
Fulham	29	29.0	48.3	29	319	2,610	29.0	36	22	58	32	4	7	69	1	35.2	29.7	21.5	51.2	498	1018	1.24	0.76	2.00	1.10	1.86	1.21	0.74	1.96	1.02	1.77
Leeds United	28	25.7	48.7	30	330	2,700	30.0	37	27	64	36	1	2	68	2	37.5	35.9	26.1	61.9	463	1153	1.23	0.90	2.13	1.20	2.10	1.25	0.87	2.12	1.20	2.06
Leicester City	28	27.0	49.3	30	330	2,700	30.0	38	28	66	37	1	2	49	2	35.6	34.1	29.5	63.6	480	1079	1.27	0.93	2.20	1.23	2.17	1.19	0.98	2.17	1.14	2.12
Liverpool	27	28.2	58.9	29	319	2,610	29.0	46	37	83	46	0	2	43	1	52.0	50.5	40.0	90.5	526	1490	1.59	1.28	2.86	1.59	2.86	1.79	1.38	3.17	1.74	3.12
Manchester City	22	28.0	64.7	29	319	2,610	29.0	73	54	127	66	7	7	36	1	61.9	56.4	44.7	101.0	759	1645	2.52	1.86	4.38	2.28	4.14	2.13	1.54	3.67	1.94	3.48
Manchester Utd	26	27.6	53.4	29	319	2,610	29.0	42	34	76	40	2	2	61	2	45.8	44.2	35.9	80.1	527	1138	1.45	1.17	2.62	1.38	2.55	1.58	1.24	2.82	1.52	2.76
Newcastle Utd	26	28.1	51.4	29	319	2,610	29.0	46	31	77	42	4	4	50	1	49.9	46.9	37.5	84.4	502	1209	1.59	1.07	2.66	1.45	2.52	1.72	1.29	3.02	1.62	2.91
Nott'ham Forest	33	27.2	40.2	30	330	2,700	30.0	23	14	37	22	1	3	70	0	30.5	28.2	22.0	50.2	371	733	0.77	0.47	1.23	0.73	1.20	1.02	0.73	1.75	0.94	1.67
Southampton	31	25.2	44.9	30	330	2,700	30.0	24	16	40	23	1	3	54	0	28.5	26.0	21.5	47.5	447	911	0.80	0.53	1.33	0.77	1.30	0.95	0.72	1.67	0.87	1.58
Tottenham	27	28.4	50.3	30	330	2,700	30.0	53	35	88	49	4	5	63	3	43.2	39.3	31.7	71.0	554	1203	1.77	1.17	2.93	1.63	2.80	1.44	1.06	2.50	1.31	2.37
West Ham	25	28.8	42.8	29	319	2,610	29.0	26	14	40	21	5	7	36	0	36.9	32.3	23.8	56.0	474	992	0.90	0.48	1.38	0.72	1.21	1.27	0.82	2.09	1.11	1.93
Wolves	31	27.4	50.5	30	330	2,700	30.0	22	10	32	20	2	2	65	6	30.0	28.5	21.4	49.9	501	1139	0.73	0.33	1.07	0.67	1.00	1.00	0.71	1.71	0.95	1.66

Figure 1 Premier League teams Standard Stats from FBref

Player Shooting 2022-2023 Premier League Share & Export ▾ Hide non-qualifiers for rate stats Glossary Toggle Per90 Stats

									Standard											Expected				
Rk	Player	Nation	Pos	Squad	Age	Born	90s	Gls ▼	Sh	SoT	SoT%	Sh/90	SoT/90	G/Sh	G/SoT	Dist	FK	PK	PKatt	xG	npxG	npxG/Sh	G-xG	np:G-xG
1	Erling Haaland	NOR	FW	Manchester City	22-266	2000	24.2	30	88	41	46.6	3.63	1.69	0.28	0.61	12.2	0	5	5	21.1	17.2	0.20	+8.9	+7.8
2	Harry Kane	ENG	FW	Tottenham	29-259	1993	29.8	23	101	41	40.6	3.39	1.37	0.19	0.46	15.7	3	4	5	17.1	13.2	0.13	+5.9	+5.8
3	Ivan Toney	ENG	FW	Brentford	27-028	1996	27.9	18	76	30	39.5	2.73	1.08	0.16	0.40	16.3	7	6	7	17.7	12.2	0.16	+0.3	-0.2
4	Marcus Rashford	ENG	FW	Manchester Utd	25-164	1997	26.5	15	79	36	45.6	2.98	1.36	0.19	0.42	15.4	4	0	0	11.8	11.8	0.15	+3.2	+3.2
5	Martinelli	BRA	FW	Arsenal	21-299	2001	27.0	14	72	27	37.5	2.66	1.00	0.19	0.52	15.6	2	0	0	8.5	8.5	0.12	+5.5	+5.5
6	Mohamed Salah	EGY	FW	Liverpool	30-302	1992	27.8	13	90	37	41.1	3.24	1.33	0.14	0.35	14.9	2	0	2	15.2	13.6	0.15	-2.2	-0.6
7	Bukayo Saka	ENG	FW	Arsenal	21-220	2001	28.0	12	67	21	31.3	2.39	0.75	0.15	0.48	16.0	0	2	2	8.8	7.4	0.11	+3.2	+2.6
8	Ollie Watkins	ENG	FW	Aston Villa	27-104	1995	26.8	12	66	37	56.1	2.46	1.38	0.17	0.30	12.5	0	1	1	12.2	11.4	0.18	-0.2	-0.4
9	Miguel Almirón	PAR	FW	Newcastle Utd	29-062	1994	22.0	11	50	17	34.0	2.28	0.77	0.22	0.65	16.7	0	0	0	6.1	6.1	0.12	+4.9	+4.9
10	Aleksandar Mitrović	SRB	FW	Fulham	28-209	1994	20.1	11	80	25	31.3	3.97	1.24	0.10	0.32	15.1	0	3	6	12.3	7.6	0.10	-1.3	+0.4

Figure 2 Premier League Players Advanced Shooting Stats from FBref

Another positive feature of FBref is their player scouting reports. FBref's player scouting reports give a snapshot of how a player performs in certain metrics. The metrics used depend on the position of the chosen player. The scouting reports give each player a percentile rank in each statistic. Players are ranked vs other players in the same position. For example, a player with a percentile rank of 99 in goals is in the top 1% of all players in that position for goals scored. These scouting reports give users an easy-to-understand summary of how good a player is and where that players strengths and weaknesses are. Alongside these scouting reports, FBref also provide similar players to the chosen player. These give users a quick comparison between players and give the players with the closest similarity based on their rank in the scouting report metrics.

FBREF Harry Kane Scouting Report

Last 365 Days View Complete Scouting Report Share & Export ▾ Glossary

vs. Forwards

Statistic	Per 90	Percentile
Non-Penalty Goals	0.49	81
Non-Penalty xG	0.40	59
Shots Total	3.37	85
Assists	0.11	43
xAG	0.16	82
npxG + xAG	0.56	69
Shot-Creating Actions	3.52	89
Passes Attempted	24.58	67
Pass Completion %	66.5%	38
Progressive Passes	4.66	96
Progressive Carries	1.65	63
Successful Take-Ons	1.16	75
Touches (Att Pen)	4.71	44
Progressive Passes Rec	6.36	53
Tackles	0.29	12
Interceptions	0.11	23
Blocks	0.60	44
Clearances	0.60	39
Aerials won	1.87	50

Similar Players to Harry Kane Last 365 Days Share & Export ▾

Forwards

Rk	Player	Nation	Squad	Compare
1	Edin Džeko	BIH	Internazionale	Compare
2	Niclas Füllkrug	GER	Werder Bremen	Compare
3	Duván Zapata	COL	Atalanta	Compare
4	Rasmus Højlund	DEN	Atalanta	Compare
5	Breel Embolo	SUI	Monaco	Compare
6	Dušan Vlahović	SRB	Juventus	Compare
7	Robert Lewandowski	POL	Barcelona	Compare
8	Romelu Lukaku	BEL	Internazionale	Compare
9	Alexis Sánchez	CHI	Marseille	Compare
10	Iago Aspas	ESP	Celta Vigo	Compare

Compare Harry Kane to Top 5 Similar Players

Similar players are based on their statistical profiles.

How similar players are calculated.

Figure 3 FBref scouting report and similar players for Harry Kane

One further positive of the FBref website is its UX. The all the stats and advanced stats come in tables. The site allows users to easily download these tables which is a nice feature for some users. These tables also allow users to easily sort by certain stats, as well as easy to use hyperlinks to other pages (for example a link to a specific team or players page). The structure of these tables also make it easy to scrape data from the FBref website.

The main limitation of FBref is its lack of visualisation. The majority of the data presented on the site is through the same data tables seen throughout the site. This makes the site easy to use but can mean that the stats can sometimes lack context. All the data is presented in raw form meaning it can be difficult the understand where players are strong and where they are not. It is here where I hope my web application can improve on this by presenting data in a more visually pleasing and easy to understand way.

3.2.1.2 WhoScored

WhoScored is a football statistics website that provides advanced statistics, match reports, previews and player ratings. Similarly to FBref, WhoScored are provided with their data from Opta.

WhoScored offer a large range of player/team stats however lack the advanced stats like xG that features in the FBref website. Like FBref, the majority of the data is presented in tables, however WhoScored does feature some other ways of visualising their data.

Player Statistics

Summary | Defensive | Offensive | Passing | Detailed

Overall | Home | Away — Minimum apps | All players

	Player	Apps	Mins	Goals	Assists	Yel	Red	SpG	PS%	AerialsWon	MotM	Rating
1	Lionel Messi PSG, 35, AM(CR),FW	25	2212	14	14	-	-	4.2	82.6	-	13	8.44
2	Martin Terrier Rennes, 26, AM(CL),FW	16	1330	9	4	-	-	2.9	75.3	3.9	7	7.71
3	Neymar PSG, 31, AM(CLR),FW	18(2)	1553	13	11	4	1	1.9	82.7	0.1	3	7.71
4	Kylian Mbappé PSG, 24, AM(LR),FW	24(2)	2102	19	3	5	-	4.2	82.9	0.3	3	7.67
5	Khvicha Kvaratskhelia Napoli, 22, AM(CLR)	24(1)	1923	12	10	-	-	2.8	82.5	0.3	7	7.61
6	Kevin De Bruyne Man City, 31, M(CLR),FW	24(3)	2124	5	14	1	-	2.2	80.1	0.3	8	7.60
7	Victor Osimhen Napoli, 24, FW	22(1)	1916	21	4	4	-	4.3	71	2	4	7.60
8	Erling Haaland Man City, 22, FW	26(1)	2189	30	5	5	-	3.4	75.6	1.3	5	7.55
9	Kieran Trippier Newcastle, 32, D(LR),M(R)	29	2558	1	6	5	-	0.4	75.1	1.4	8	7.54
10	Robert Lewandowski Barcelona, 34, FW	23(1)	1952	17	6	1	1	3.9	78.6	1.7	7	7.54

© WhoScored — Page 1/155 | Showing 1 - 10 of 1543 first | prev | next | last

Mins: Minutes played — *Goals: Total goals* — *Assists: Total assists*
Yel: Yellow card — *Red: Red card* — *SpG: Shots per game*
PS%: Pass success percentage — *AerialsWon: Aerial duels won per game* — *MotM: Man of the match*
** Only players from English Premier League, French Ligue 1, German Bundesliga, Italian Serie A and Spanish La Liga are displayed*

Figure 4 WhoScored Player Data Table

One of the unique features of WhoScored is their ratings. Each player is given a rating for each match out of 10. The ratings are calculated by a WhoScored model which features a large variety of player data from the match like shots, passes, tackles, etc. These can be used to compare different players and teams across different leagues and competitions. They also provide users with an objective measure of a player or team's performance, and give the data context. Another unique feature of WhoScored is their Characteristics. These are characteristics of a player or team which describe what a player or teams strengths and weaknesses are, as well as their style of play. These are calculated vs the league averages for specific team or player statistics.

Figure 5 WhoScored ratings for the Premier League this season

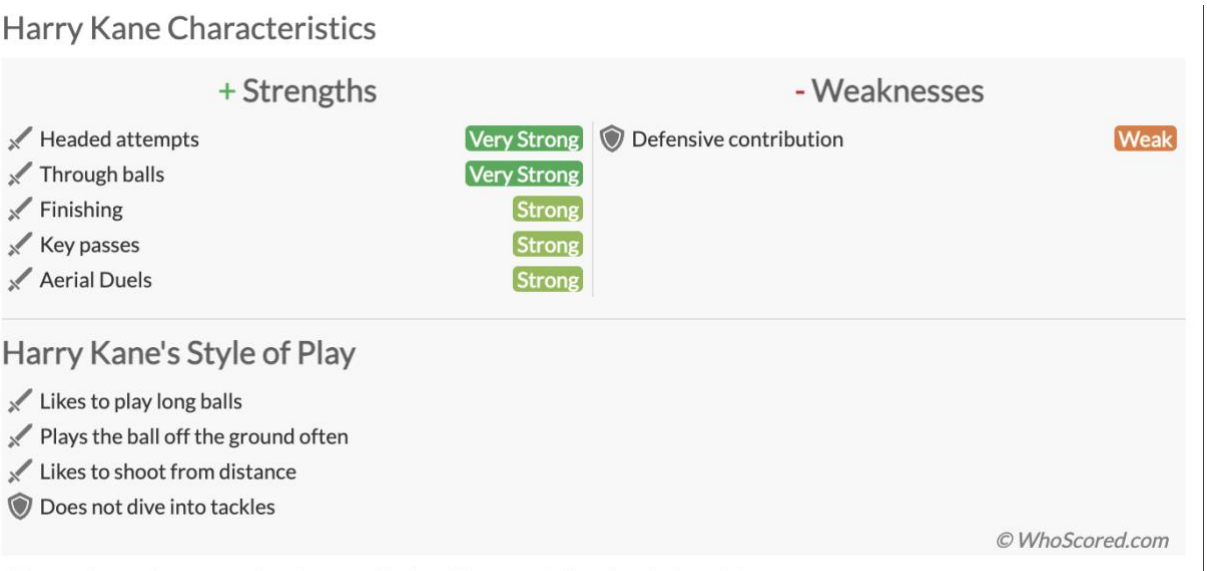
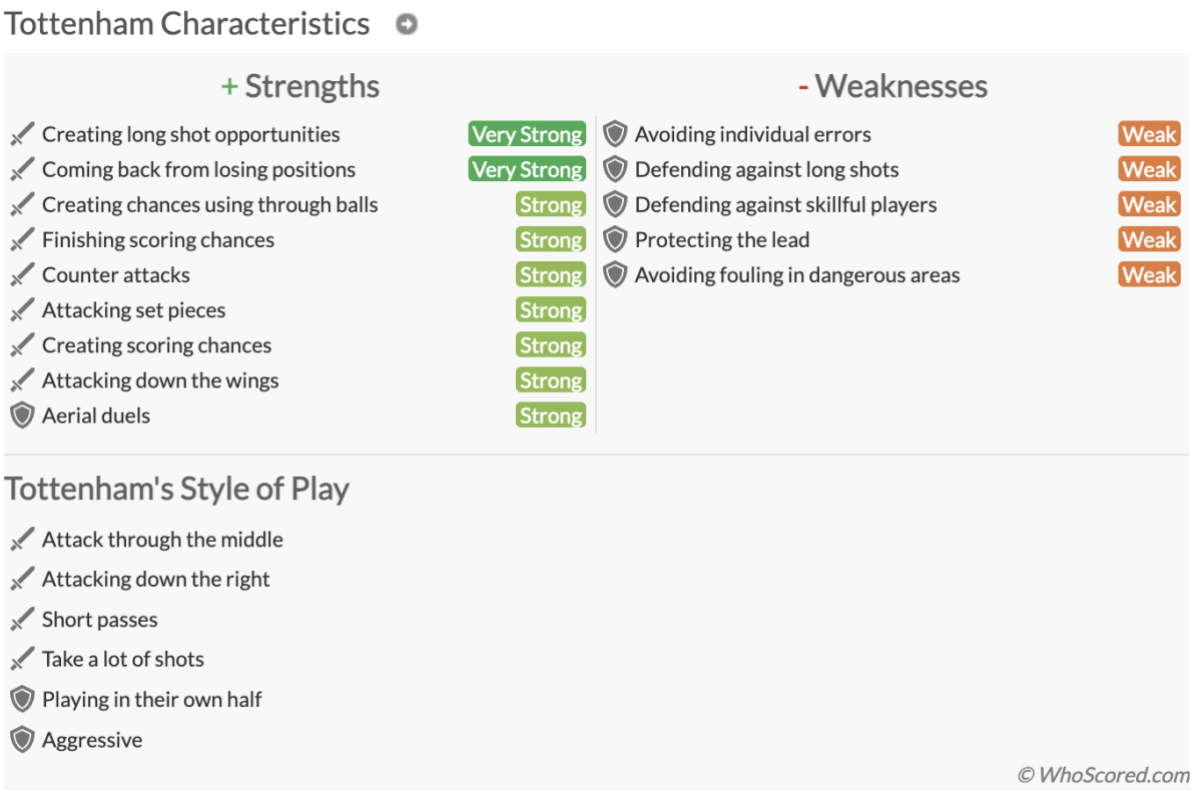


Figure 6 Player Characteristics for Harry Kane



** Strengths, weaknesses and styles are calculated from statistics of each player's latest two seasons*

Figure 7 Team Characteristics for Tottenham

Although WhoScored has positive features like their ratings and characteristics. Aside from these features the website is lacking in terms of the data available of FBref and lacks the user interface of FBref.

3.2.1.3 Data Scouting (https://unarsezer.shinyapps.io/data_scouting/)

Data Scouting is an R Shiny app developed by Sezer Unar. The site allows users to select a players and then displays data about that player. This site has a number of good features, one being the player roles. This assigns each player to a role based on the type of player they are. These roles are based on a model. These roles give the user a good understanding of what sort of player the selected player is. The site also features a description of each role so it's clear what the players role means.

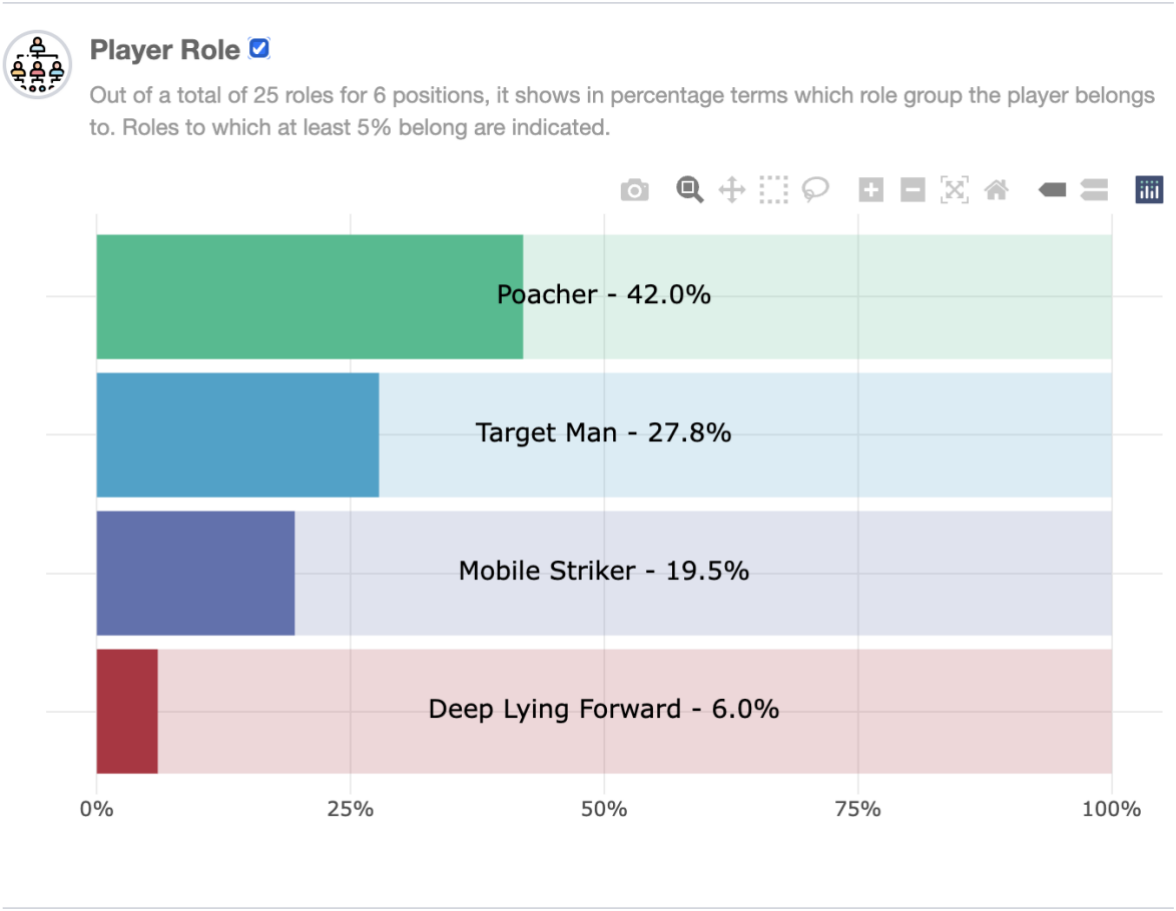


Figure 8 Example of Player Roles for Harry Kane

Another good feature of this site is the playstyle feature. This gives each player a percentile rank in a number of statistical categories to give a good understanding of what a player is good out compared to others. This data is displayed in a radar chart. One problem with this feature is that the chart has the same statistics for all players without taking into account the position of the player. For example this is a problem because the statistics that are relevant to a forward are less relevant to a defender.

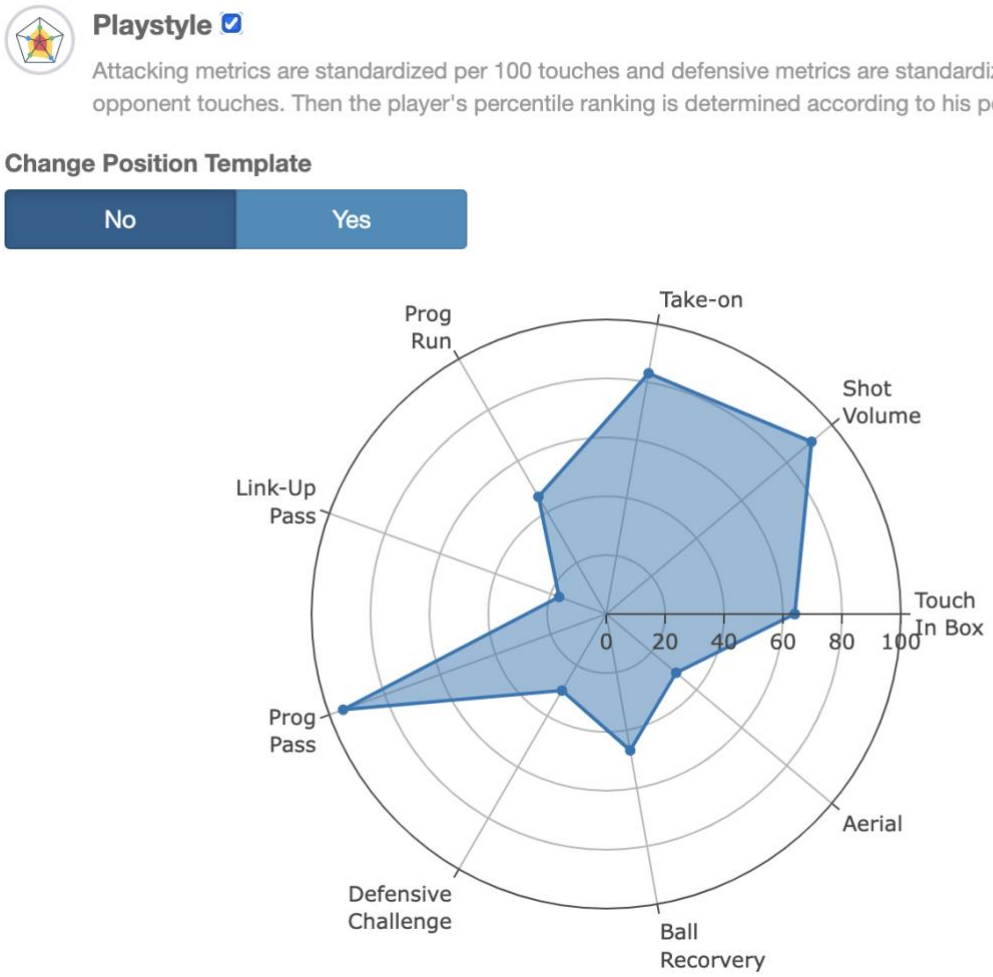


Figure 9 Playstyle radar chart for Harry Kane

Finally, another positive feature in this site is the compare player chart. This displays a scatter plot with the user selecting the selecting the statistics that go on the x and y axis. The selected player is then highlighted on this scatter plot with the plot containing all the players in the database. The plot also allow the user to select the leagues and the positions that feature in the plot. One way to this feature could be improved would be with more filtering available to the user as well as more statistics available to be compared.

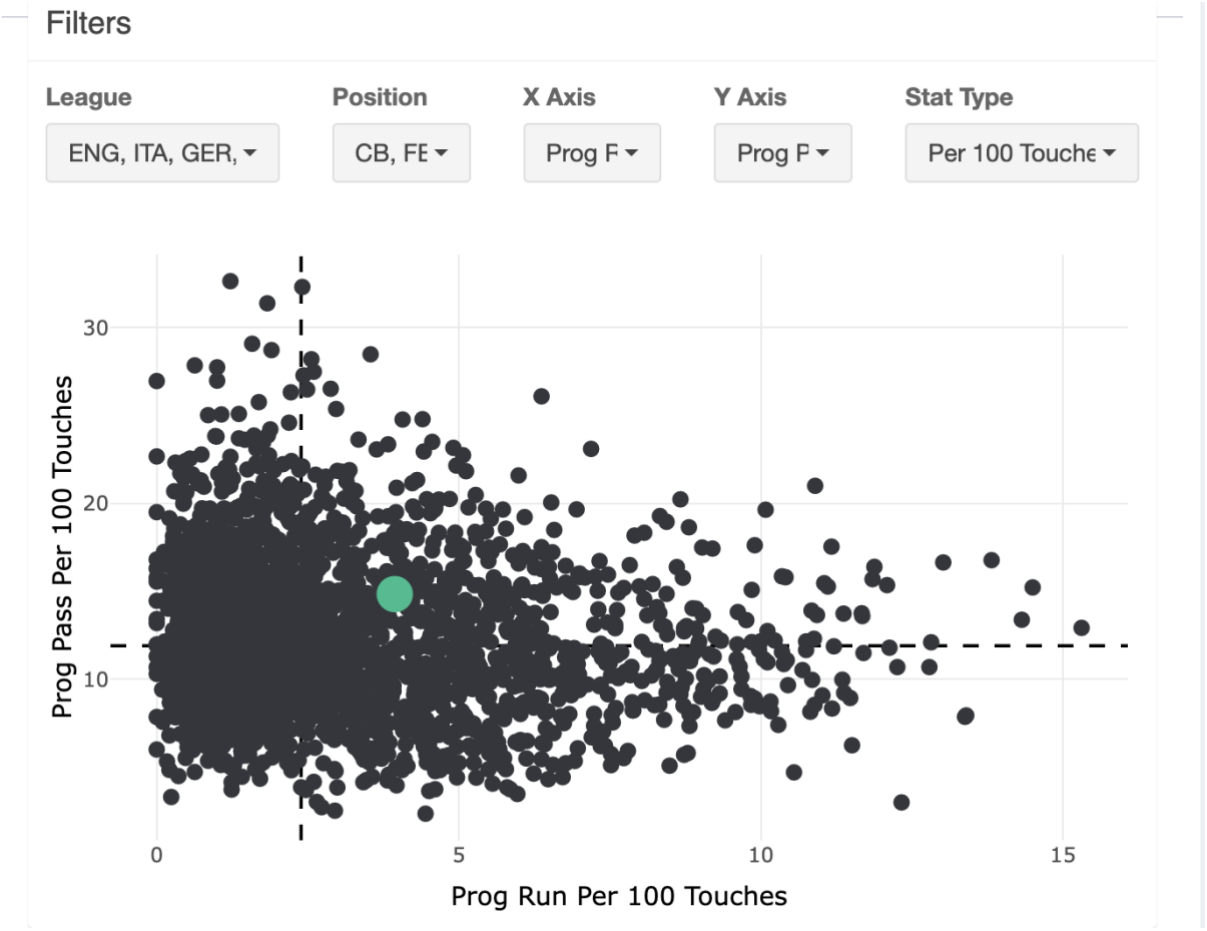


Figure 10 Scatter Plot comparing players from Data Scouting Site

Finally another con of this site is that it only features data from this year. Unlike other sites it doesn't allow users to view data from previous or historic season. Seeing data from historic season would allow users to be able view trends in the data as well as comparing players or teams from different years.

3.2.2 Survey

An important step in the requirements gathering is conducting a survey with potential users. This gives an opportunity to get opinions from potential users on what they would like from the application. The purpose of the survey was to find out whether this application would be in demand and what features would be popular among users.

The results of the survey were mostly positive. It was clear from the results of the survey that there was a clear interest in users for an application like this. All the users that took part in the survey categorised themselves as football fans. The majority of these said that they were interested in how data is used in football and see data visualisation as a valuable tool to judge player performance. Participants were the asked what leagues they watched most often. The large majority watched the English premier league, with a few also watching some other leagues in Europe. This is important

because it helps outline the data needed for the application. No participants responded that they watched leagues outside the top 5 leagues, this shows that the data needed should focus on these leagues. When asked about the type of data that potential users would be looking for in the application, the majority were interested in player level data, while a couple also were interested in team level data. No participants were interested in league level data.

Overall the results of the survey with potential users match what my expectations for the application were. This is why performing this survey is important because sometimes as developers we can ignore what potential users want and only focus on our initial concept.

3.3 Requirements modelling

3.3.1 Personas

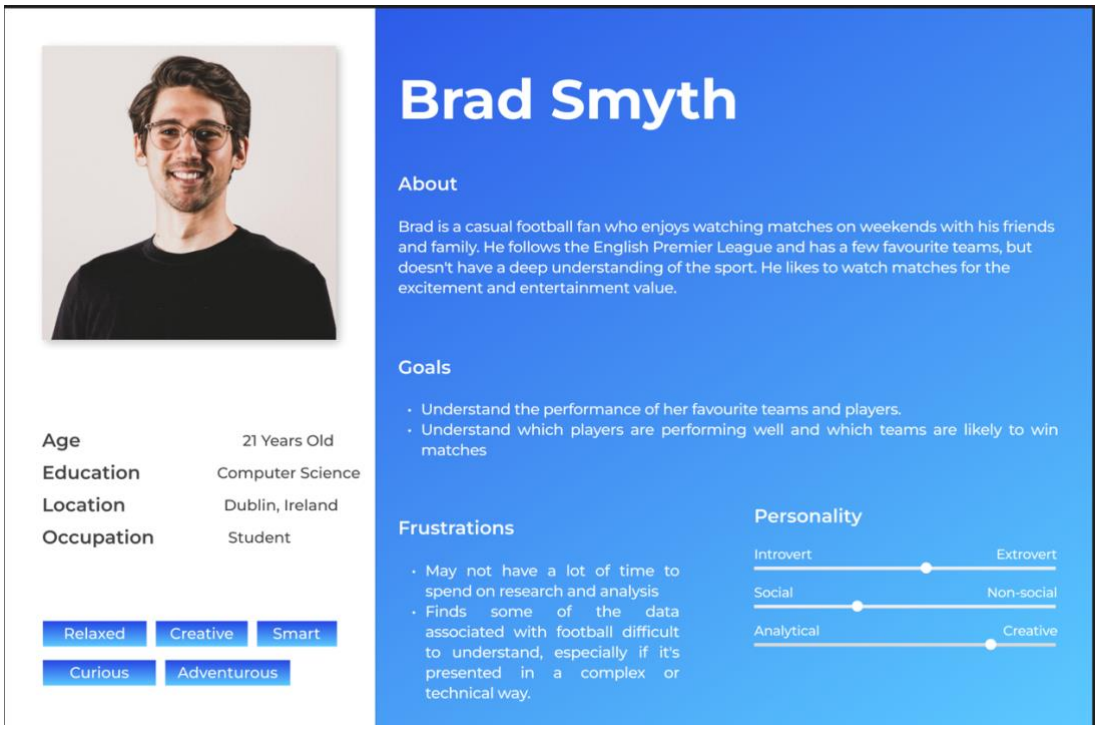


Figure 11 Persona for casual football fan

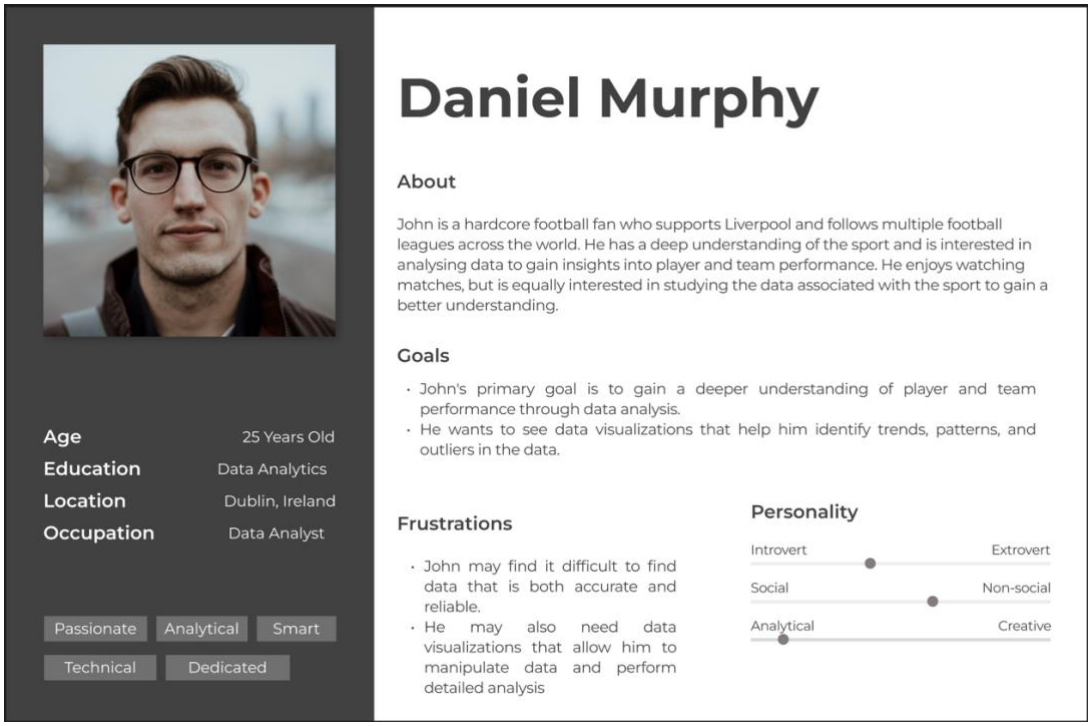


Figure 12 Persona for football fan with analytical interest

3.3.2 Functional requirements

1. The web application must scrape data to be used in the application.
2. Display a scatter plot of the player data.
3. Allow users to filter the data through a number of different user inputs like league, position.
4. Allow users to interact with the plot.
5. Highlight a specific player in the plot.
6. Show the same data in a data table.
7. Allow users to download the data from the data table.
8. Allow users to save the plot as an image.
9. Display player percentile ranks in certain statistics in a pizza chart.
10. Allow user to select a player and display data on that player in a pizza chart.
11. Allow user to select a second player to compare players.
12. Allow users to download the pizza charts as images.

3.3.3 Non-functional requirements

- **User Interface:** The application should have a user friendly interface to allow users easily navigate the site.
- **Compatibility:** The application should be compatible on different screen sizes like laptops and mobiles for example.
- **Scalable:** The application should be able to handle large amounts of data and users.

- **Performance:** The application and its data should load in a sufficiently quick amount of time.
- **Up to Date:** The data being used in the application should be as up to date as possible.

3.3.4 Use Case Diagrams

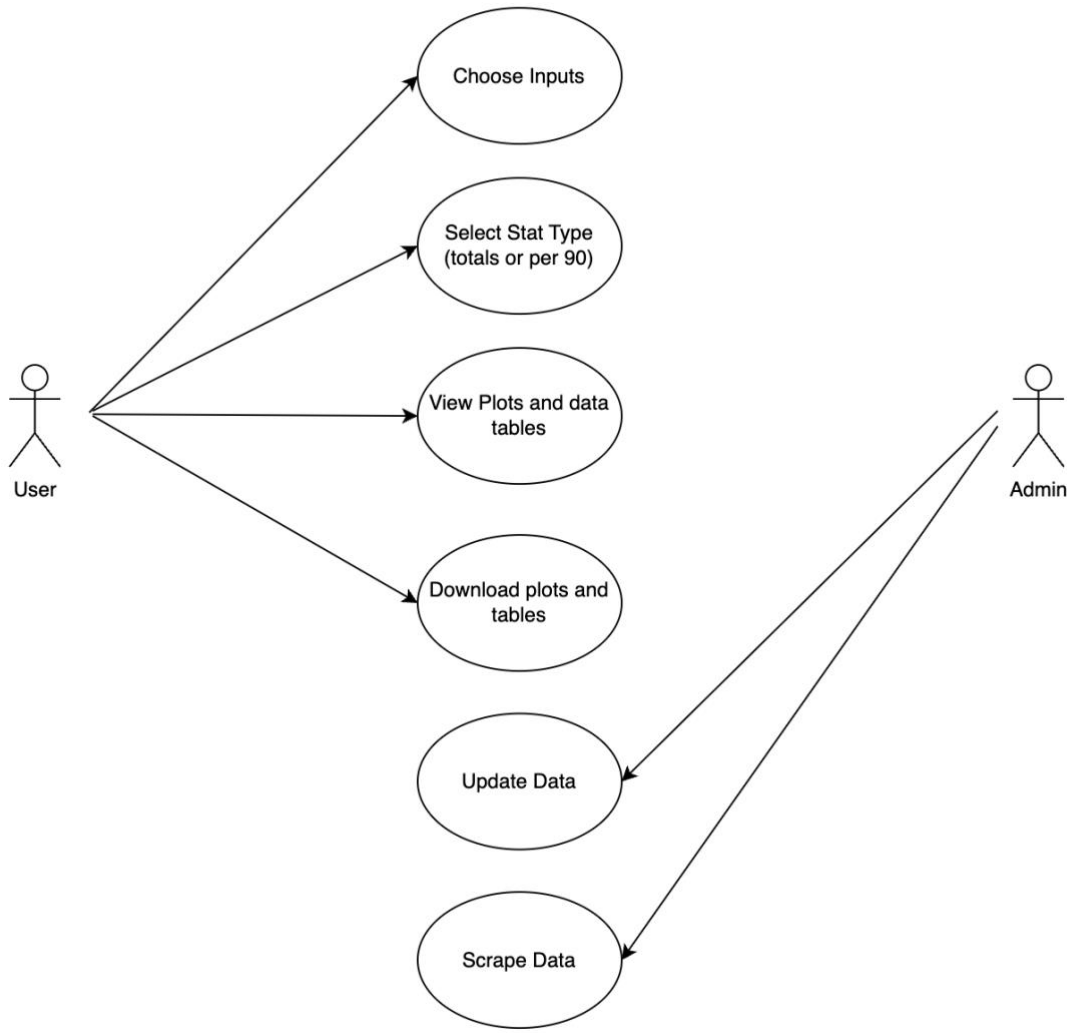


Figure 13 Use Case Diagram

3.4 Feasibility

The two main technologies that will be used in the development of this web application is R and Shiny. R is a programming language used for statistics and graphics. It is popular or its ability to handle statistical computations and visualisations. One of the key strengths of R is its ability to produce high-quality graphics and visualizations, which can be customized to suit the needs of the developer. R provides a range of packages for creating graphics, including ggplot2, which provides a

flexible and powerful system for creating data visualizations, Plotly, which provides interactive graphics, and the dpylr package, which provides functions for data manipulation. R can also be used for statistical modelling and machine learning. One of the main R packages that will be used in the development of the application is the worldfootballR package. This is a R package that provides predefined functions to scrape sites like FBref. This package will be vital for sourcing the data needed for this application because FBref does not have an API available. This package also contains functions for pre-scraped data which speeds up the process of getting the data.

Shiny is a web application for R that allows developers to build interactive web applications using R. Shiny allows developer to build web applications without the need for HTML, CSS or JavaScript, however they can also be used alongside it for improvement if needed. Shiny makes use of reactive programming which allows the application to dynamically update based on input from the user. The user interface of a Shiny application can be customized using a variety of widgets and controls. These controls can include sliders, inputs, etc. When a user uses these widgets, Shiny updates the visualization based on the updated input. This allows the development of responsive and interactive applications that explore large and complex datasets. Like R, Shiny also has a vast number of packages that can help aid and improve the development.

The use of R and Shiny will cause no compatibility issues because the two technologies were developed to be used alongside each other. Shiny is developed to use the R programming language as well as its packages and libraries to create interactive web applications.

3.5 Conclusion

This chapter helps the outline what the application should be able to do. This is done by first researching similar applications and identifying the positive and negative features of these sites. A survey was then conducted with potential users to gain an understanding of what users would want from the application and what features would get users to use the application. The results of this survey were in line with my opinion for what the application should be.

Having gathered this information, it was used to outline the functional and non-functional requirements of the application. The functional requirements outline what the application should be able to do. The non-functional requirements focus on the user experience of the application.

4 Design

4.1 Introduction

Having gathered the requirements in the previous chapter it was then possible to outline the design for the application. The design for the application can be divided into two sections: the program design and the user interface design. The aim of this chapter is to develop a design for the application to match the requirements outlined in the requirements chapter.

The program design relates to the design of the code of the application. By outlining this in the design phase it makes the coding of the application easier.

The user interface design outlines what the front end of the application will look like. This is done through the development of prototypes and wireframes which outline the appearance of the application.

4.2 Program Design

4.2.1 Technologies

The main technologies that will be used in the development of this application is R and Shiny. R is a programming language for statistical and graphical computing. Shiny is a web application framework for R that allows for the development of interactive web applications using R code. The combination of these technologies allow for the development of important platforms or tools for data analysis and data visualisation.

R provides a large range of tools for data manipulation, processing and visualisation. R contains a wide variety of packages, libraries and functions which aid in a variety of different tasks. The main reason for choosing R is its capabilities in handling data manipulation, processing and visualisation for large amounts of data. One important R package is the tidyverse package. This is a core package that contains a number of different packages. These include dpylr and ggplot2. Dpylr is useful for the manipulation and pre-processing data to suit the developer's needs. Ggplot2 is an important package for data visualisation.

Shiny allows for the development of interactive web applications. Shiny provides a set of functions to create user interfaces using R. Shiny helps build these user interfaces using templates and widgets which can be updated using HTML,CSS and JavaScript however does not require the use of these languages. Shiny is built on the reactive programming model which allows the application to update dynamically based on user input. This allows the development of applications that give users the ability to explore large datasets as well as the ability to visualise these dynamic datasets.

Alternatives to use in the development of this application could have been JavaScript. JavaScript is a programming language popular in the development of web applications. Like R JavaScript has a large variety of libraries many of which can be used for data visualisation, such as D3.js and Charts.js. However JavaScript does not offer the same level of statistical computation and visualisation that R

does so it wasn't used in the development of the project. Also, a lot of the JavaScript libraries that would be used have equivalents that can be used with R and Shiny.

Another alternate that could have been used is Python and the Dash framework. Like R and JavaScript discussed above, Python has a variety of libraries that aid in the development of data visualisation web applications. However it was decided that R and Shiny are a better stack to use for this project.

4.2.2 Structure of R Shiny

A Shiny application consists of two main parts: The UI and the Server function.

The user interface function is the function that defines what the user sees and interacts with. It uses a variety of R and Shiny functions to create various types of inputs, like select inputs and sliders, and outputs, like tables and plots.

The server function is the backend of the application. It receives the values from the UI function and processes them and then returns the output to the UI function. The server and the UI function are connected by reactive expressions. These are functions that dynamically update the server whenever the inputs in the UI function change.

For the sake of this application the structure will be a one file application. Meaning the UI and server functions will both be in the one file. It is possible to have these in two separate files.

```
library(shiny)

# Define the UI
ui <- fluidPage(
  # UI elements go here
)

# Define the server function
server <- function(input, output) {
  # Server logic goes here
}

# Run the application
shinyApp(ui, server)
```

Figure 14 Basic Structure of a Shiny application

Outside of the application file, there is the data folder. The data folder contains the R files for scraping and pre-processing the data to be used in the application. Once the data is scraped and processed using R code the data is then saved as a csv file to be used in the Shiny application.

Figure 15 Structure of data folder - contains R files and CSV files

4.2.3 Design Patterns

The design pattern used for the coding of this web application is the Model-View-Controller design pattern. In this design pattern the applications code is divided into three different components. By separating the application into different components it allows the code to be modified or extended independent of each other. This can be beneficial for scalability and debugging purposes. By separating the code it makes it easier to write additional code and better understand existing code.

The model component is responsible for handling the data for the application. For the sake of this project the model is responsible for the data scraping and pre-processing/manipulation. The model in the application is the two R files found in the data folder of the source code.

The view component is responsible for the user interface of the application. In this case the view component refers to the UI function of the application. It is responsible for the R and Shiny Ui inputs and widgets, the outputs like plots and tables and any additional UI code that may be used in the application like HTML,CSS and JavaScript.

The controller component is the server function of the application. It is responsible for handling the user input from the View component. In this application this makes use of reactive programming, whereby changes to the inputs automatically update the UI outputs.

4.2.4 Application architecture

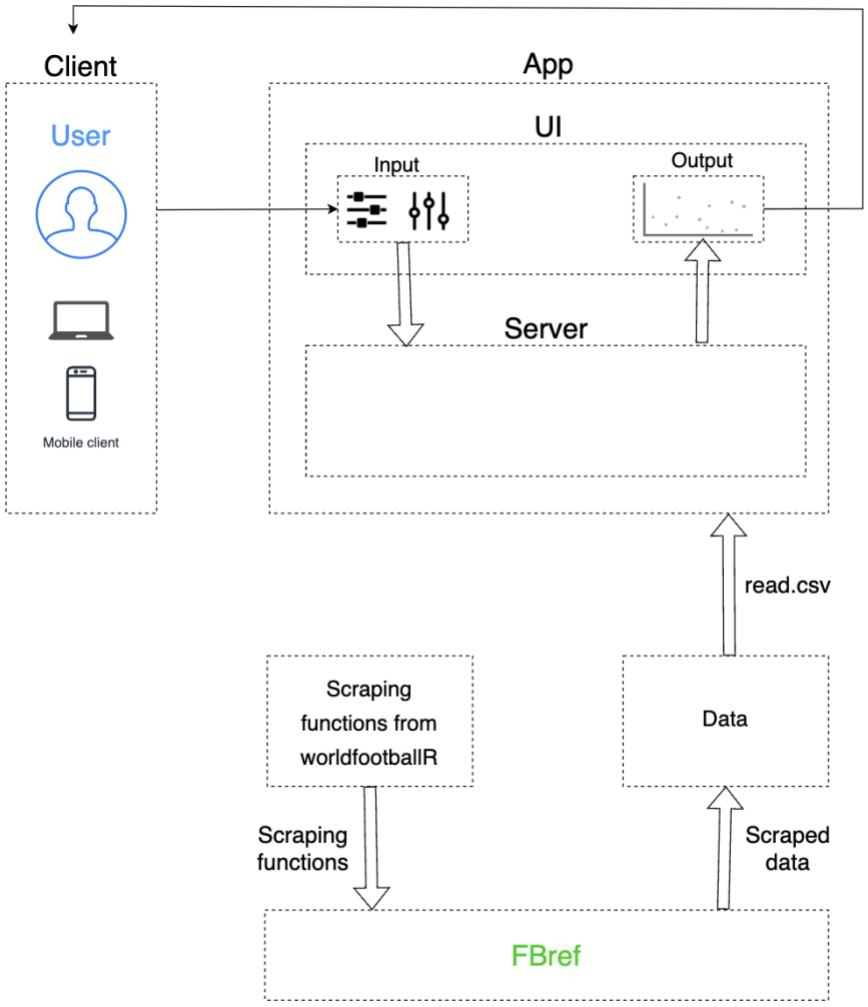


Figure 16 Block Diagram of application architecture

4.2.5 Process design

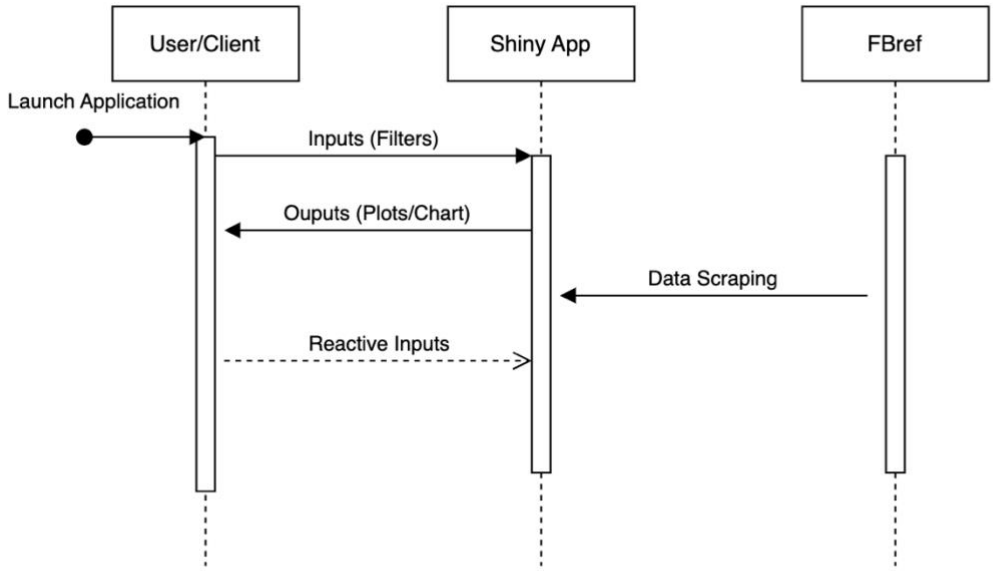


Figure 17 Sequence Diagram

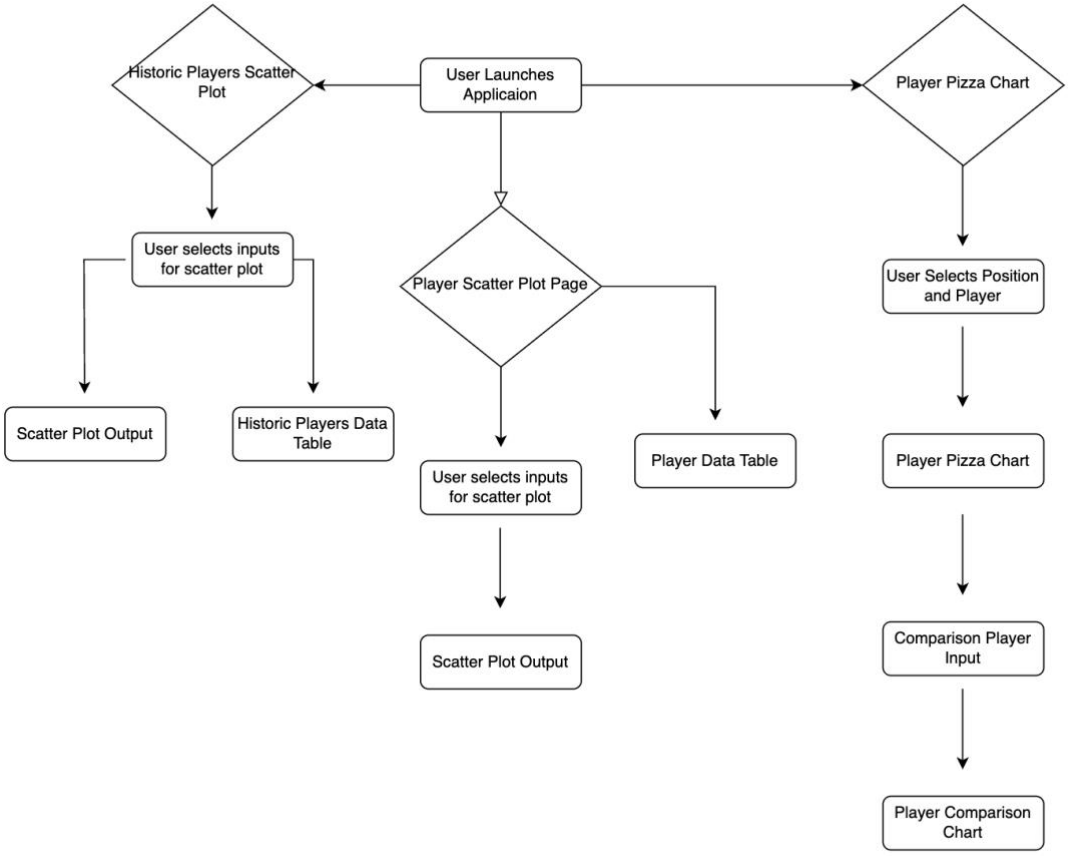


Figure 18 Flow Chart

4.3 User interface design

4.3.1 Paper Prototypes

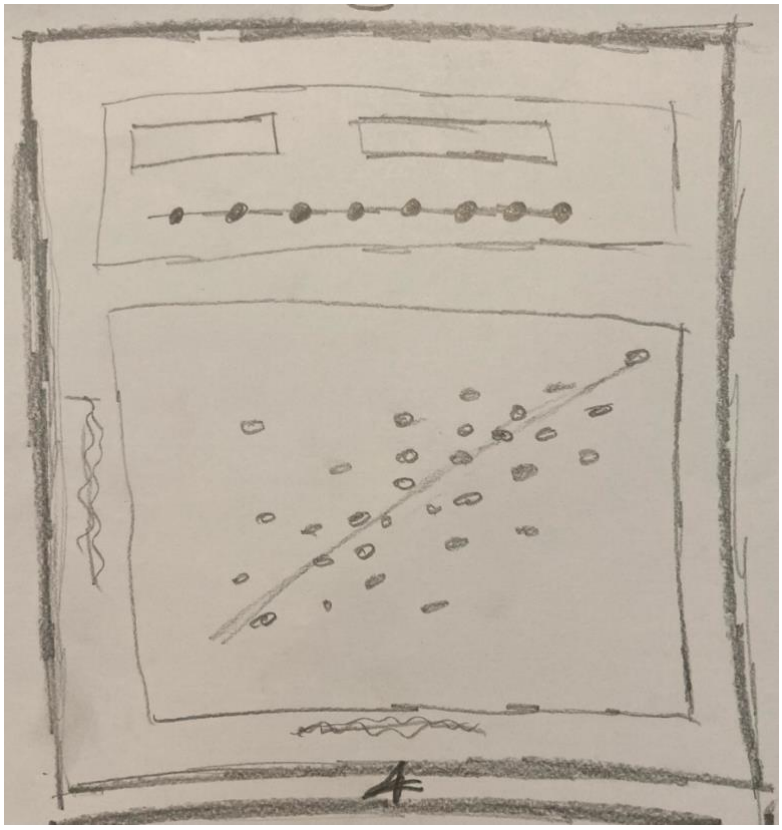


Figure 19 Sketch of scatter plot page

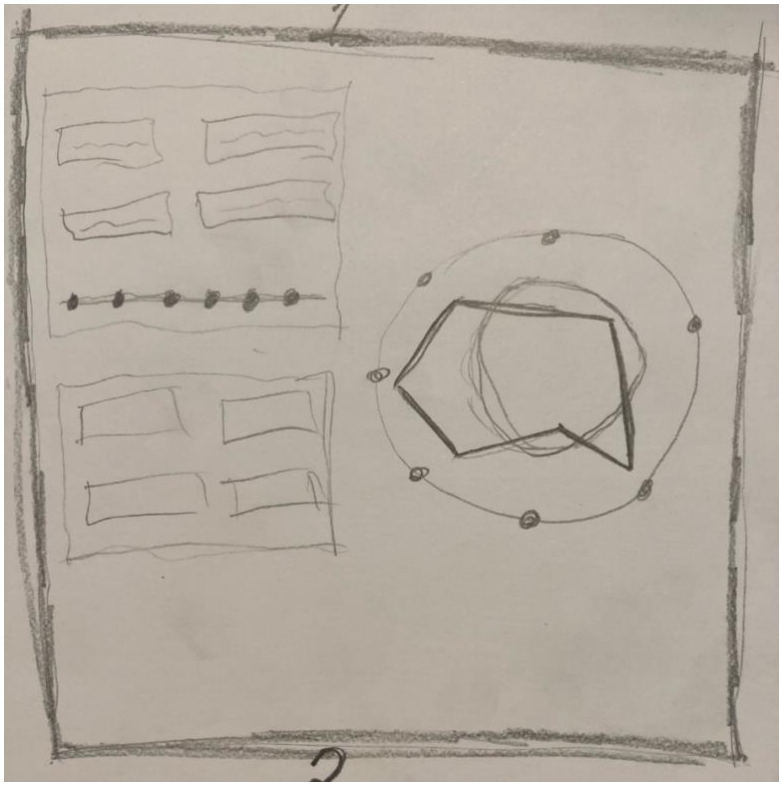


Figure 20 Sketch of player pizza chart page

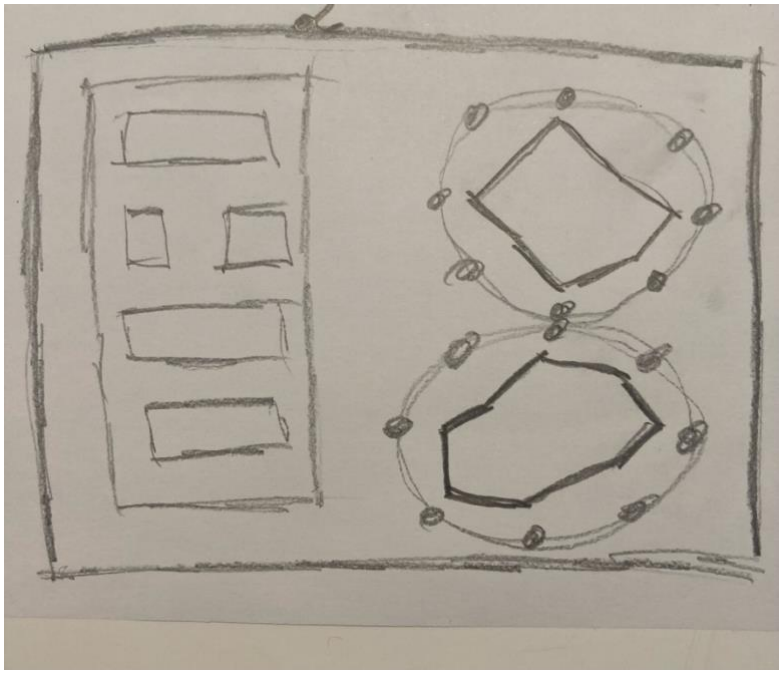


Figure 21 Sketch of Pizza chart comparison

4.3.2 Wireframe

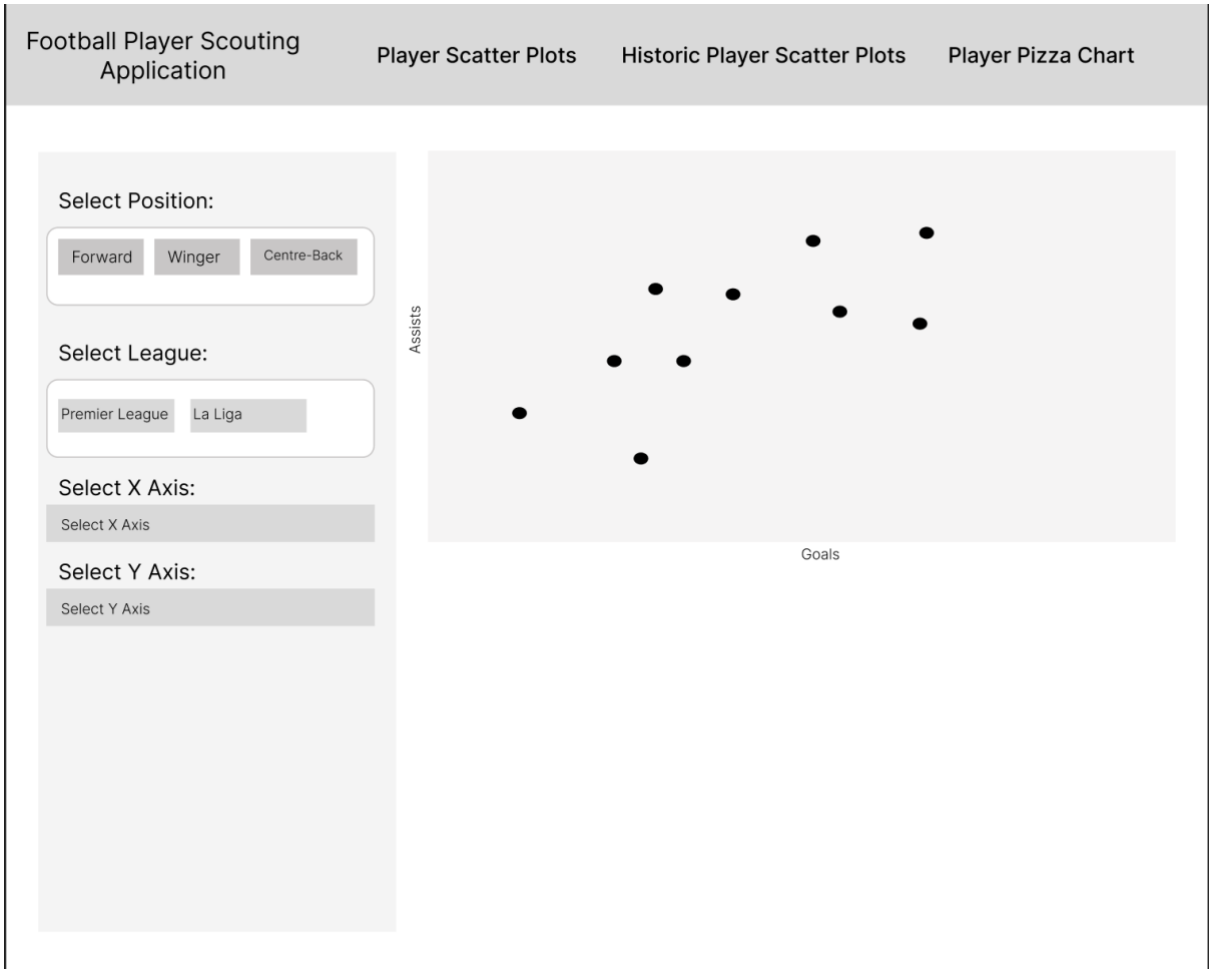


Figure 22 Scatter Plot Page Wireframe

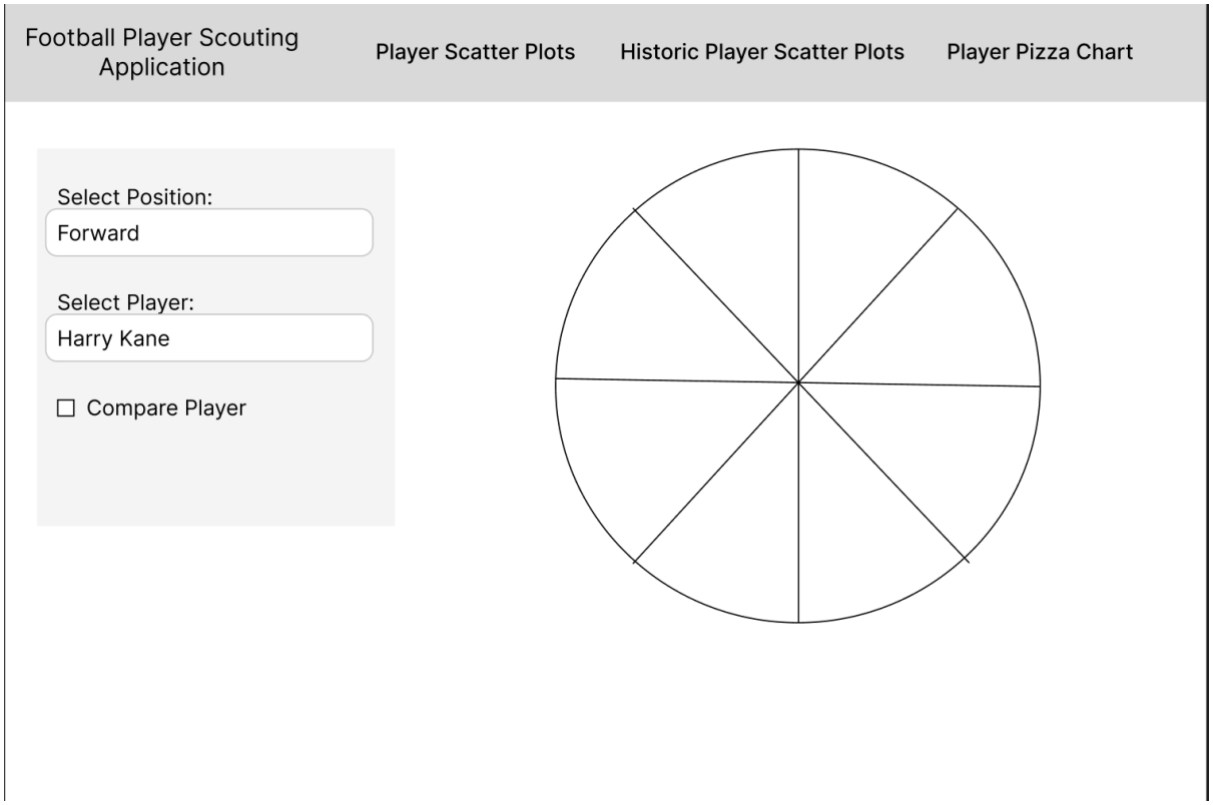


Figure 23 Pizza Chart Wireframe

4.4 Conclusion

This chapter outlines the design of this project. This was split into two different sections, the program design and the UI design. The process design outlines the design of the code. This helps with the implementation of the project. In this section I outline the technologies that will be used in the development of the application, and discuss the structure of the R Shiny code and the design pattern used. The program design is concluded with a sequence diagram and a flow chart.

The UI design section outlines how the application will look. This is done with the paper prototypes that were designed at the very beginning of the project when the initial concept was being developed. Figma is then used to create wireframes showing the UI of the application.

5 Implementation

5.1 Introduction

This chapter outlines the implementation of the web application. The goal of this chapter was to develop a web application using R and Shiny that matches the application outlined in the requirements and design chapters. The project was implemented using the Scrum Methodology which involved breaking down the project a series of sprints. During each sprint different features and functionalities for the application were implemented. The application was developed using R and Shiny. R is a statistical programming language that is popular for data analysis, modelling and visualisation. Shiny is a R web application framework for building interactive data visualisation applications using R.

The goal of this project was the develop a football data visualisation web application. This application will scrape data to be used, and then display the data in a series of visualisations for users to view on football players. The idea is for the application to be a tool for fans/analysts to scout football players through the use of data.

5.2 Scrum Methodology

The Scrum Methodology was used during the development of this project. It is an Agile framework used for project management during a software development process. The Scrum methodology is about teamwork and collaboration. It involves taking an incremental approach to development, by breaking the project down into smaller "sprints". Each sprint lasts two weeks, with each one having a goal set for the end, i.e. a feature/functionality to be implemented. The goals/tasks outlined at the start of each sprint are known as the backlog. During each sprint, meetings take place to discuss progress and encourage team work and collaboration. For the sake of this project each sprint involved meeting with my supervisor to keep track of progress in each sprint.

One of the main benefits of using the Scrum methodology is the flexibility it gives. By breaking down the project into sprints it means the developer can focus on a functionality or feature one at a time which can make the development and debugging process easier.

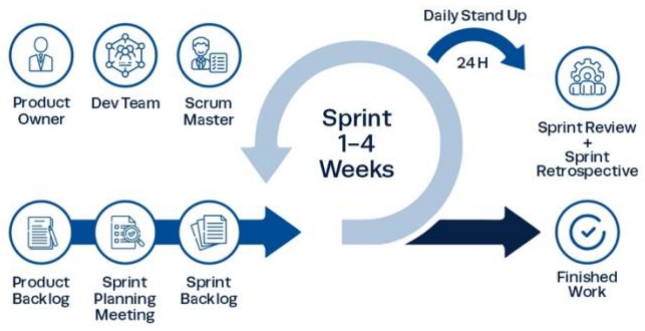


Figure 24 Scrum Methodology

In conclusion, the use of Scrum Methodology was an important part of this project by making the development process more efficient and easier to manage. The focus on teamwork and collaboration in Scrum is another important part of the process by allowing for feedback during the sprints.

5.3 Development environment

The IDE (Integrated Development Environment) used in the development of this project was R Studio. R Studio has a user friendly interface to write, debug and test code for data analysis and visualisation. The R Studio IDE provides multiple panels that provide different functions to aid in the development of R and Shiny applications. The first panel is where all the coding happens. The code is written and run in this panel. The second panel contains the console and terminal. The next panel has a view of all the data frames that are being used in the project. Finally the fourth R Studio panel contains a few important features. These include a tab to install and load different R packages, a tab to view any R visualisations, a tab showing the files in the working directory, and finally a help tab that provides help on different R packages.

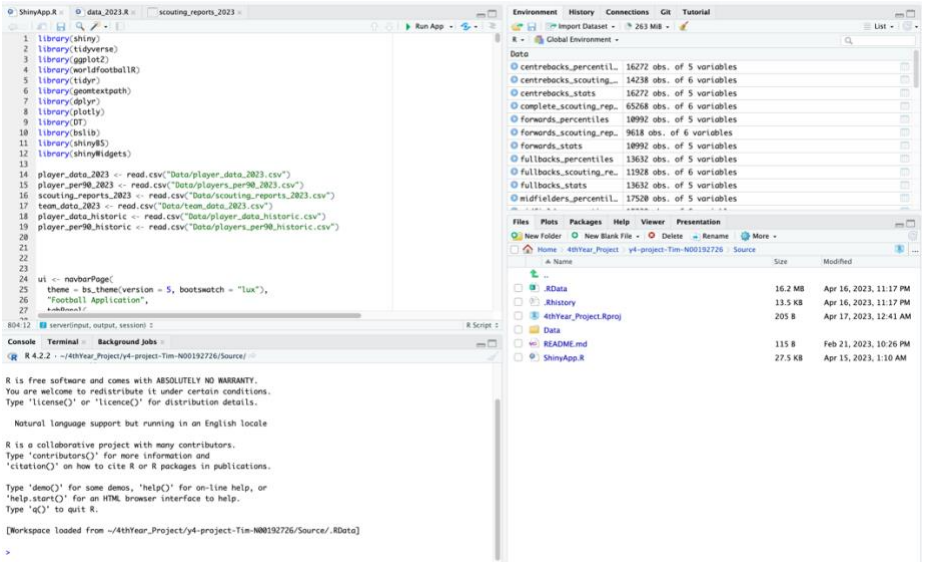


Figure 25 R Studio IDE

As well as the R Studio IDE GitHub was another important tool used in the development of this Shiny application. GitHub provided version control during the development process. This allowed me to easily keep track of changes during the development. By using GitHub to keep track of changes it allows me to keep track of where bugs or issues may have occurred. GitHub desktop was used to easily commit to GitHub. GitHub desktop made it very quick and easy to commit any code to GitHub without the need to even use the command line or terminal.

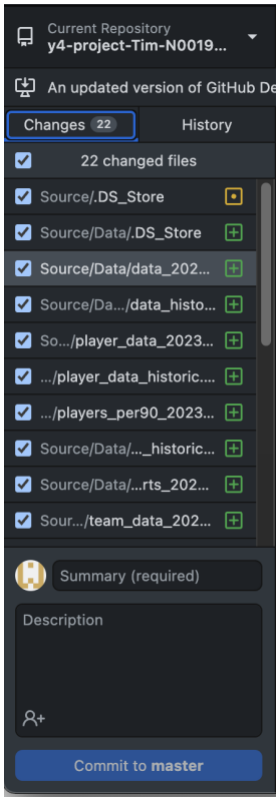


Figure 26 GitHub Desktop Interface

5.4 Data

5.4.1 Data Scraping

The data used in this project is from the website FBref. FBref is a free website that provides advanced football data. FBref is supplied with all their data by Opta Sports. FBref gives permission to scrape their data, however you can be blocked for making more than one request every three seconds. It is also important to cite FBref when using their data which will be done in this application.

To scrape data from FBref I used an R package called worldfootballR. This was an important and useful package in the development of this project. WorldfootballR provides predefined functions for scraping data from a number of sites including FBref, as well as Transfermarkt which is also used during the development of this application. This package allowed me to scrape data with only a couple of lines of code.

One important worldfootballR function used was the fb_big5_advanced_season_stats() function. This returned a data frame of all data for all players in the top 5 leagues (England, France, Italy, Germany and Spain). This is the main data used in the application. The functions takes in a number of arguments for what data is returned, season_end_year, stat_type and team_or_player. The season_end_year argument decides the year of the data returned, for example giving it a value of 2023 returns data for the 2022/2023 season. The stat_type argument defines the stat type that is

returned. These include Standard, Shooting, Possession, Passing, Defence. Finally the team_or_player decides whether the data is player data or team data.

```
20
21  player_standard <- fb_big5_advanced_season_stats(
22    season_end_year = 2023,
23    stat_type = "standard",
24    team_or_player = "player"
25  )
```

Figure 27 Function to return Standard Player stats for 2022/2023

	Season_End_Year	Squad	Comp	Player	Nation	Pos	Age	Born	MP_Playing
1	2023	Ajaccio	Ligue 1	Mickaël Alphonse	GLP	DF	33	1989	23
2	2023	Ajaccio	Ligue 1	Cédric Avinel	GLP	DF	36	1986	21
3	2023	Ajaccio	Ligue 1	Mickaël Barreto	FRA	MF	32	1991	15
4	2023	Ajaccio	Ligue 1	Cyrille Bayala	BFA	MF	26	1996	28
5	2023	Ajaccio	Ligue 1	Youcef Belaïli	ALG	MF,FW	31	1992	16
6	2023	Ajaccio	Ligue 1	Kouamé Botué	BFA	FW,MF	20	2002	8
7	2023	Ajaccio	Ligue 1	Florian Chabrolle	FRA	MF	25	1998	6
8	2023	Ajaccio	Ligue 1	Yanis Cimignani	FRA	MF,DF	21	2002	8
9	2023	Ajaccio	Ligue 1	Mathieu Coutadeur	FRA	MF	37	1986	21
10	2023	Ajaccio	Ligue 1	Ismael Diallo	CIV	DF	26	1997	24

Figure 28 Example of Standard Player data frame for 2022/2023

Figure 22 shows the function to get the standard player data for the 2022/2023 season. This data frame contains more columns then shown in the above figure. This data is then assigned to a data frame called player_standard. This is done for each stat type: Standard, Shooting, Passing, Possession, GCA, Defense and Misc.

The data scraped in this project is split into two files. The data scraping discussed above is all for the 2022/2023 season. The other data scraped is historical data. The worldfootballR package also has pre-scraped that can be scraped using load_fb_big5_advanced_season_stats(). This function is the same as the fb_big5_advanced_season_stats() function, however, instead of scraping the data from FBref it is retrieved from a repository in the worldfootballR GitHub. This speeds up the length of time it takes to scrape multiple seasons at a time. This “load” function takes in the same arguments as the fb_big5_advanced_season_stats() function. It allows the season_end_year argument to be multiple years. FBref only have advanced statistics dating back to the 2017/2018 season so that is as far back as the historical data goes.

```r
player_standard <- load_fb_big5_advanced_season_stats(
  season_end_year = 2018:2022,
  stat_type = "standard",
  team_or_player = "player"
)
```

Figure 29 Function to return historical player data from 2017-2022

The figure above shows the function to scrape the standard player data from the 2017/2018 season to the 2021/2022 season. The colon is used for selecting multiple season in the season_end_year argument. Like the data for the 2022/2023 season this is done for all the stat types (i.e. standard, passing, shooting, etc.

	Season_End_Year	Squad	Comp	Player	Nation	Pos	Age	Born	MP_Playing
	All	All	All	Harr	All	All	All	All	All
2409	2018	Tottenham	Premier League	Harry Kane	ENG	FW	24	1993	37
5072	2019	Tottenham	Premier League	Harry Kane	ENG	FW	25	1993	28
7766	2020	Tottenham	Premier League	Harry Kane	ENG	FW	26	1993	29
10603	2021	Tottenham	Premier League	Harry Kane	ENG	FW	27	1993	35
13503	2022	Tottenham	Premier League	Harry Kane	ENG	FW	28	1993	37

Figure 30 Example of Historical Data for Harry Kane

5.4.2 Data Processing/Manipulation

5.4.2.1 Player Scatter Plot Data

Although the scraper functions provided by worldfootballR made it efficient to scrape the data for the project, the data required a large amount of data processing and manipulation to suit the needs of the application. This processing/manipulation was a large and important part of the implementation process.

As discussed in the data scraping section above the data is scraped by the statistic type, for example standard, shooting, passing, etc. Each stat type scraped has to be processed so that the data and the data frame is in a state that can be best used in the shiny application.

The first thing done to the data is to index the data frames of each stat type. This is done by the line of code below:

```r
30  player_standard$index <- 1:nrow(player_standard)
```

Figure 31 indexing stat type data frame

This code adds a new column to the player_standard data frame (or the equivalent in another stat type, i.e. player_shooting) called “index”. This index column is assigned a number, which gives each row a unique identifier. The “nrow” function gets the length of the data frame which allows an index value to be assigned to each row.

The next thing done to process the data is to select the desired player statistics from each stat type. When the data is scraped the data frames contain a large number of columns, each a different advanced player statistic type like goals, assists, progressive passes, etc. However, not every statistic is needed for the application, so the code below is used to filter only the columns needed.

```
player_standard_filtered <- player_standard |>
  select(
    index,
    Squad,
    Comp,
    Player,
    Age,
    Url,
    Nation,
    Pos,
    Starts_Playing,
    MP_Playing,
    Min_Playing,
    Gls,
    Ast,
    G_minus_PK,
    npxG_Expected,
    xAG_Expected,
    PrgC_Progression,
    PrgP_Progression,
    PrgR_Progression
  )
```

Figure 32 selecting columns to be used

This code shows the player_standard data frame being filtered using the select function and then assigning the selected columns to a new data frame called player_standard_filtered. This is done for all the stat type data frames, like player_standard, player_shooting, etc. as seen below.

Data	
player_defense	2796 obs. of 27 variables
player_defense_filter…	2796 obs. of 8 variables
player_gsca	2796 obs. of 27 variables
player_gsca_filtered	2796 obs. of 3 variables
player_misc	2796 obs. of 27 variables
player_misc_filtered	2796 obs. of 4 variables
player_passing	2796 obs. of 34 variables
player_passing_filter…	2796 obs. of 7 variables
player_possession	2796 obs. of 33 variables
player_possession_fil…	2796 obs. of 9 variables
player_shooting	2796 obs. of 28 variables
player_shooting_filte…	2796 obs. of 6 variables
player_standard	2796 obs. of 39 variables
player_standard_filte…	2796 obs. of 19 variables

Figure 33 filtered data frames for each stat type

When filtering the columns of each data frame it was important that the index column was selected for each stat type data frame.

Having filtered each data frame to contain the desired columns, the data frame were then combined together into one data frame. This is done by creating a list that contains the data frames and then using the reduce() function to combine the data frames in the list.

```
150
151  player_combined_df = list(
152    player_standard_filtered,
153    player_shooting_filtered,
154    player_passing_filtered,
155    player_gsca_filtered,
156    player_defense_filtered,
157    player_possession_filtered,
158    player_misc_filtered
159  )
160
161  player_combined_df <- player_combined_df |>
162    reduce(right_join, by = 'index')
```

Figure 34 Combining data frames into player_combined_df

The right_join argument combines the desired data frames to the right of each other. He “by” argument then specifies the column used to combine the data frames, in this case the index column that was previously added to each data frame.

Having combined the data frames into one singular data frame the next thing done during the data processing was to improve the players positions values. When the player data is scraped from FBref using the worldfootballR package the players have general positions like defender, midfielder, and forward. However, for this application it was important that the players are given their more specific position like Full-Back, Centre-Back, Winger, etc. This is so that players can be compared with players in the same specific position. For example, instead of all defenders being compared together, they are now split into full-backs and centre-backs because full-backs have different roles to centre-backs.

The player_dictionary_mapping() function from worldfootballR is used to get these more specific positions. These function is used to join player data from FBref and Transfermarkt. Transfermarkt is another football data website. It is popular for providing evaluations of players prices. For this project Transfermarkt is only used to provide the positions for players.

```
164  player_combined_df <- player_combined_df |>
165    rename(Name = Player) |>
166    rename(UrlFBref = Url)
167  transfermarkt_players <- player_dictionary_mapping()
168  transfermarkt_players <- transfermarkt_players |>
169    select(-PlayerFBref)
```

Figure 35 player_dictionary_mapping function to get player positions

In this code above the data from player_dictionary_mapping() is stored in the transfermarkt_players data frame. This code also shows some data processing done on the player_combined_df data frame and the transfermarkt_players data frame. In the player_combined_df data frame the Player and Url columns are renamed to Name and UrlFBref. The Url column is changed name to join the data frames as discussed below. In the transfermarkt_players data frame the PlayerFBref column is dropped. This is to avoid having two columns for the player name when the data frames are combined together.

```
171  player_positions = list(transfermarkt_players, player_combined_df)
172  player_combined_df <- player_positions |>
173    reduce(right_join, by = 'UrlFBref')
```

Figure 36 combining data frames for positions

The above code shows the two data frames being combined. This is done in the same way as when joining the stat type data frames. In this case the two data frames are joined by the FBref URL. This means that the values in the transfermarkt_players data frame are combined with the data in the

player_combined_df data frame where the FBref URL is the same. At first I was joining by the Name columns however this caused issues where players have the same name. By using the FBref Url this fixed the issues because the URL is unique to each player. From this resulting data frame the old Position column is then dropped along with the Transfermarkt URL because they are no longer needed.

Some of the positions got from the methods discussed above were specific to the side the player plays on. For example Left or Right Backs, or Left or Right midfield. For this application it is not necessary for the positions to be side specific. To fix this some positions were combined together into one like below.

```
player_combined_df$Position <-
  ifelse(
    player_combined_df$Position == "Left-Back" |
      player_combined_df$Position == "Right-Back",
    "Full-Back",
    player_combined_df$Position
  )
```

Figure 37 Combining left and right backs to full-backs

In this code, rows with values “Left-Back” or “Right-Back” are changed to “Full-Back”. This is also done for some other positions in the same way as seen below.

```
player_combined_df$Position <-
  ifelse(
    player_combined_df$Position == "Left-Back" |
      player_combined_df$Position == "Right-Back",
    "Full-Back",
    player_combined_df$Position
  )
player_combined_df$Position <-
  ifelse(
    player_combined_df$Position == "Central Midfield" |
      player_combined_df$Position == "Defensive Midfield",
    "Midfielder",
    player_combined_df$Position
  )
player_combined_df$Position <-
  ifelse(
    player_combined_df$Position == "Centre-Forward" |
      player_combined_df$Position == "Second Striker",
    "Forward",
    player_combined_df$Position
  )
player_combined_df$Position <-
  ifelse(
    player_combined_df$Position == "Left Winger" |
      player_combined_df$Position == "Right Winger",
    "Winger",
    player_combined_df$Position
  )
player_combined_df$Position <-
  ifelse(
    player_combined_df$Position == "Left Midfield" |
      player_combined_df$Position == "Right Midfield",
    "Winger",
    player_combined_df$Position
  )
```

Figure 38 Changing positions

Having got the player positions that are required this data frame has been completed. A separate data frame is also created from the player_combined_df for per 90 data. The per 90 data is calculated by dividing each stat by the amount of minutes played by the player and then multiplying by 90. The data is then saved and exported as a csv file that can be imported into our application code. These data frames are used for the player scatter plots in the application.

```
player_data_2023 <- player_combined_df

write.csv(player_data_2023, "Data/player_data_2023.csv")

players_per90_2023 <- player_combined_df |>
  mutate_at(vars(12:50), ~ (. / Min_Playing) * 90) |>
  mutate_at(vars(12:50), round, 2)

write.csv(players_per90_2023, "Data/players_per90_2023.csv")
```

Figure 39 Creating per90 data frame and exporting both data frames

The data processing for the historic data is done in the exact same way as has been discussed above. The data frames for the historic data are also similarly exported to CVSs to be used in the application.

5.4.2.2 Player Pizza Chart Data

The code discussed in the previous section was to get the data to be used in the player scatter plots in the application. However, that data is also used as the start point for processing the data for the player pizza charts. Specifically the per 90 data frame. For the player pizza charts we need to get the players percentile rank in each statistic. This is the rank a player is in a certain statistic compared to other players in the same position. For example a player in the 99 percentile for goals is in the top 1% of players in the same position for goals scored.

The processing done for the player pizza charts is done by position. The positions are split up into different data frames before being combined together. This is so that the players can only be compared against others in the same position. Within each position group the data frames are dived into two data frames: percentile and stats. The stats data frame contains the raw per 90 values for each statistic, while the percentile data frames contain the percentile values for a player in each statistic.

```
231  forwards_stats <- players_per90_2023 |>
232    filter(Position == "Forward") |>
233    filter(Min_Playing >= 450) |>
234    gather(Statistic, Value,-Name,-Position)
```

Figure 40 Stats data frame for Forwards

```
236  forwards_percentiles <- players_per90_2023 |>
237    filter(Position == "Forward") |>
238    filter(Min_Playing >= 450) |>
239    mutate(across(where(is.numeric), ~ round(cume_dist(.), 2))) |>
240    gather(Statistic, percentile,-Name,-Position)
```

Figure 41 Percentiles data frame for Forwards

To process these data frames to suit the needs of the pizza charts, there is first some filtering applied to them. This includes filtering to the position of the data frame, in this case forwards. Then we filter to only players that have played a minimum of 450 mins. This is so that outliers in the per 90 numbers are not included (players who have played little minutes that may skew the percentiles). The gather function is then used to convert the data frame to long form. In this new data frame the original columns are rows in a new “Statistic” column and the old values of each column are now in the “Value” column. The other two columns in the new forward_stats data frame are the player name and position. By using the minus before the column name in the gather function the column is

not added to the long form of the data frame. The resulting forward_stats data frame can be seen below.

	Position	Name	Statistic	Value
	All	Kane	All	All
1	Forward	Harry Kane	Url	https://fbref.com/en/players/21a66f6a/Harry-Kane
2	Forward	Harry Kane	index	2467
3	Forward	Harry Kane	Squad	Tottenham
4	Forward	Harry Kane	Comp	Premier League
5	Forward	Harry Kane	Age	29
6	Forward	Harry Kane	Nation	ENG
7	Forward	Harry Kane	Starts_Playing	31
8	Forward	Harry Kane	MP_Playing	31
9	Forward	Harry Kane	Min_Playing	2775
10	Forward	Harry Kane	Gls	0.75
11	Forward	Harry Kane	Ast	0.06
12	Forward	Harry Kane	G_minus_PK	0.62
13	Forward	Harry Kane	npxG_Expected	0.46
14	Forward	Harry Kane	xAG_Expected	0.16
15	Forward	Harry Kane	PrgC_Progression	1.62
16	Forward	Harry Kane	PrgP_Progression	4.67
17	Forward	Harry Kane	PrgR_Progression	5.51
18	Forward	Harry Kane	Sh_Standard	3.47

Showing 1 to 18 of 48 entries, 4 total columns (filtered from 11,952 total entries)

Figure 42 Example of the long form format of the forward_stats data frame

The processing for the forwards_percentiles data frame is the same as the for the forward_stats data frame with one difference. The percentile values are calculated to replace the per 90 data. This is done using the cume_dist() function. This function ranks the data by counting the number of rows with a value of less than or equal to the given value and dividing it by the number of entries. A number of other functions are used for implementing the percentiles. The across() function is used to apply the function to multiple columns and the where() function decides which columns to apply it. In this case all columns where the value is numeric. Finally the round() function is used to round the percentile value to 2 decimal places. These calculations are done before the data frame is converted to long form in the same way as the forward_stats data frame. With the new columns being called “Statistic” like the above and “percentile” instead of value. An example of the percentiles can be seen below.

	Position	Name	Statistic	percentile
	All	All	Gls	All
1	Forward	Erling Haaland	Gls	1
2	Forward	Victor Osimhen	Gls	1
3	Forward	Kylian Mbappé	Gls	0.99
4	Forward	Wissam Ben Yedder	Gls	0.99
5	Forward	Alexander Isak	Gls	0.98
6	Forward	Cristhian Stuani	Gls	0.98
7	Forward	Eric Maxim Choupo-Moting	Gls	0.98
8	Forward	Karim Benzema	Gls	0.97
9	Forward	Terem Moffi	Gls	0.97
10	Forward	Christopher Nkunku	Gls	0.96
11	Forward	Daniel Ciofani	Gls	0.96
12	Forward	Harry Kane	Gls	0.96
13	Forward	Robert Lewandowski	Gls	0.96
14	Forward	Alexandre Lacazette	Gls	0.95
15	Forward	Marcus Ingvartsen	Gls	0.95
16	Forward	Jonathan David	Gls	0.94
17	Forward	Roberto Firmino	Gls	0.94
18	Forward	Andy Delort	Gls	0.93

Showing 1 to 18 of 249 entries, 4 total columns (filtered from 11,952 total entries)

Figure 43 forwards_percentiles data frame showing ranks for Goals scored

Having calculated the percentile values for each player the forward_percentiles and forward_stats data frames are assigned an index and then joined together using this index. This resulting data frame is called forward_scouting_reports. Finally, some of the non-statistic entries are dropped from the Statistics column like Age, URL, Squad, Nation, etc.

```
242 forwards_stats$index <- 1:nrow(forwards_stats)
243 forwards_percentiles$index <- 1:nrow(forwards_percentiles)
244
245 forwards_scouting_reports <-
246   merge(forwards_stats, forwards_percentiles[, c("percentile", "index")], by = "index")
247
248 forwards_scouting_reports <-
249   filter(
250     forwards_scouting_reports,
251     Statistic != "index",
252     Statistic != "Age",
253     Statistic != "Squad",
254     Statistic != "Comp",
255     Statistic != "Nation",
256     Statistic != "Url"
257   )
```

Figure 44 combining data frames and dropping values not needed

This same process is then done in the same way for the other positions like Centre-Backs , Full-Backs, etc. The only difference being that Attacking Midfielders and Wingers are both done together so that the positions can be compared and ranked with each other. The data frames are then combined together using the rbind() function as seen below.

```
372 complete_scouting_reports <-
373   rbind(
374     forwards_scouting_reports,
375     wingers_attmids_scouting_reports,
376     midfielders_scouting_reports,
377     centrebacks_scouting_reports,
378     fullbacks_scouting_reports
379   )
380
381 complete_scouting_reports$percentile <-
382   as.numeric(complete_scouting_reports$percentile)
383 complete_scouting_reports$Value <-
384   as.numeric(complete_scouting_reports$Value)
385 class(complete_scouting_reports$Value)
386
387
388 complete_scouting_reports$percentile <-
389   complete_scouting_reports$percentile * 100
390
391 scouting_reports_2023 <- complete_scouting_reports
392
393 write.csv(scouting_reports_2023, "Data/scouting_reports_2023.csv")
```

Figure 45 combining all position scouting reports and final processing

The final processing done to this data is after they have been combined together. The code on lines 381 to 385 above is converting the values in the percentile and Value columns to be numeric. Previously they were actually characters which was causing a bug in the rankings. The values in the percentile column are then multiplies by 100 to give them a percentage value instead of decimal. Finally, the data frame is saved and exported to a Csv to be used in the application.

5.5 Application

The application code starts by importing the packages to be used in the application. These packages are very important for implementing this project. The first package imported is the Shiny package. This packages enables shiny in the r code to be used. This provides functions to help build the UI, like inputs and outputs, and reactive functions for the server which is the shiny apps back-end.

Another package that is imported is the tidyverse package. This package contains a number of other packages like dpylr, tidyr and gglot2. The dpylr package is the same as the package used in the data processing scripts. The ggplot2 package is used for implementing the plots for the application. Two other packages imported are the plotly and DT packages. The plotly package is used to make the scatter plots interactive. The DT package is used for the data tables used in the application. Finally,

the bslip package is imported to provide themes for the user interface of the application. Having imported the packages, the data is then imported from the r data files discussed previously.

```
1  library(shiny)
2  library(tidyverse)
3  library(worldfootballR)
4  library(plotly)
5  library(DT)
6  library(bslib)
7  library(geomtextpath)
```

Figure 46 importing packages to shiny application

```
9  player_data_2023 <- read.csv("Data/player_data_2023.csv")
10 player_per90_2023 <- read.csv("Data/players_per90_2023.csv")
11 scouting_reports_2023 <- read.csv("Data/scouting_reports_2023.csv")
12 team_data_2023 <- read.csv("Data/team_data_2023.csv")
13 player_data_historic <- read.csv("Data/player_data_historic.csv")
14 player_per90_historic <- read.csv("Data/players_per90_historic.csv")
```

Figure 47 importing data to shiny application

The shiny application code is split into two sections, the UI and Server functions. The UI function contains all the elements that the user sees, like the input and outputs. The server function has the code that generates the outputs that are displayed in the UI. The server function also contains the interactive logic for the application. It contains the reactive functions that controls how to outputs are updated based on the inputs.

5.5.1 Player Scatter Plots

The original idea for the player scatter plots function was to allow the users to select the data that is displayed by allowing the user to select the column from the data frame that is displayed on the X and Y axis. Each column is a different player statistic. Throughout the process of developing the project more and more features and functionality have been added or changed to improve the scatter plots. In this section I will discuss the implementation of the scatter plot feature.

5.5.1.1 UI Layout

The shiny layout functions help structure the user interface of the application. There are a number of different type of layout functions to use in Shiny.

```
17  ui <- navbarPage(
18    theme = bs_theme(version = 4, bootswatch = "litera"),
19    "Football Data Scouting Application",
20    tabPanel(
21      "Player Scatter Plots",
22      sidebarLayout(
23        sidebarPanel(
```

Figure 48 Ui layout of application

The navbarPage() function is the top level layout function of the Shiny application. It provides a navigation bar at the top of the page which is then toggles between using the tabPanel() function. Each tabPanel() is a separate page. Within each tab panel there is then another layer of layout functions to define the UI of each page. In this application each page is a sidebarLayout(). The sidebarLayout() is made up of the sidebarPanel() function and the mainPanel() function. The side bar makes up 1/3rd of the page and contains the inputs. The main panel makes up the other 2/3rd of the page. This is where the outputs are displayed.

5.5.1.2 X and Y Axis Inputs

The select inputs for the x and y axis are quite simple. Within each select input is three arguments. The input ID is defined first and is used to link between the input on the UI function and the plot in the server function. The second variable is the label of the input. Finally, the inputs choices are set. In the case of the x and y axis inputs the choices are the columns in the player data frame.

```
41          selectInput("x", "Select x axis:",
42                      c(colnames(player_data_2023)),
43                      selected = ""),
44          selectInput("y", "Select y axis:",
45                      c(colnames(player_data_2023)),
46                      selected = ""),
```

Figure 49 Select Inputs for selecting columns displayed on X and Y axis

This code shows the X axis input being defined as x, with the Y axis input being set as y. The columns used in the input is then set using the colnames() function which returns all the columns from the player_data_2023 data frame. The user interface this code generates can be seen below.

Figure 50 X and Y Axis Select Input

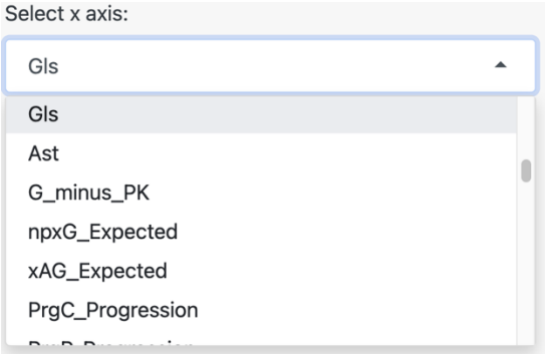


Figure 51 Select Input options for X and Y Axis

5.5.1.3 Position and League Inputs

```
24          selectizeInput(
25            "Positions",
26            "Select Positions:",
27            choices = unique(scouting_reports_2023$Position),
28            selected = c(scouting_reports_2023$Position),
29            multiple = TRUE,
30            options = list(plugins = list('remove_button'))
31          ),
32          selectizeInput(
33            "Comp",
34            "Select League:",
35            choices = unique(player_data_2023$Comp),
36            selected = c(player_data_2023$Comp),
37            multiple = TRUE,
38            options = list(plugins = list('remove_button'))
39
40          ),
```

Figure 52 Position and League Inputs

The inputs for the position and league uses the selectizeInput() function. The selectize input allows for some extra features from selectize.js. In these inputs they take in an input ID like previous, in this case “Positions” for the position input and “Comp” for the league input.

The choices value for positions input contains the values in the scouting_reports_2023 data frame. By using the unique() function it only shows each position once, which speeds up the loading time of the page. Previously when not using the unique() the input would go through the entire data frame which slowed down the load time significantly. The reason for using the scouting_reports_2023 data frame is because the player_data_2023 data frame contains some NULL values in the position column. By setting the selected value to be the same as the choices value it means that all the positions or leagues are selected originally. These inputs are able to have multiple values by setting the multiple value to true. Finally, the options value is the reason for using the selectize input. It allows us the use the remove_button() function from selectize.js. This provides a remove icon to each value in the input when selected.

Figure 53 Position and League UI Inputs

5.5.1.4 Statistic Type Radio Button

The next UI input added to the scatter plot was the statistic type. This allowed the user to select whether the player data displayed is “totals” or “per 90” values. Per 90 numbers are an important feature to have because they allow users to few the quality of players who may have played less minutes then others. The total data favours those who have played the most minutes, by using per 90 we can better compare player who have played less with players who have played a lot based on their quality with the same amount of time played. The inputs for this can only be one of two values so the best input type to use was a radio button. Like the other inputs the radio button inputs contains an ID that is used in the server, as well as the choices that are displayed in the radio button. In this case the choices are “Total” and “Per 90”.

```
47          radioButtons("radio", "Stat Type:",
48                       choices = c("Totals", "Per 90s")),
```

Figure 54 code for stat type radio button

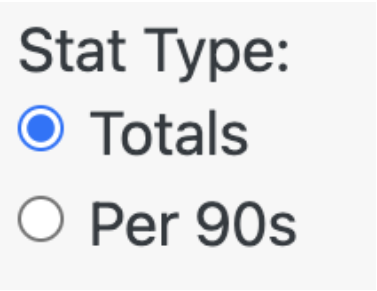


Figure 55 Stat Type Radio Button

5.5.1.5 Age and Minutes Sliders

The next UI element implemented in the application were two slider inputs for Minutes and Age. The minutes slider was the first to be implemented. This was an important input to have once the radio button for totals and per 90 stat was added. The per 90 data can contain many outliers in the data from players playing low amount of minutes. By adding a minutes slider it allows the user to filter the data that is being displayed by amount of minutes. This allows them to raise the minimum minutes to reduce the outliers who have player little amount of minutes.

```
49          sliderInput(
50            inputId = "slider",
51            label = "Minutes",
52            min = 1,
53            max = max(player_data_2023$Min_Playing),
54            value = c(450, max(player_data_2023$Min_Playing))
55          ),
```

Figure 56 Minutes slider input

Like all the other inputs, the slider input has an ID that will be used in the server function. The slider input also has two arguments for Min and Max. These are the minimum and maximum values on the slider. For the slider input the minimum is one minute played and the maximum is equal to the highest amount of minutes played in the player_data_2023 data frame. This is done using the max() function. The value argument is used to set the initial value of the slider. In this case the set low value is set at 450 minutes. This is a good value to remove outliers who have played low minutes and filters to players who's data is an accurate representation of that player.

The Age slider was added later in the development process. It is a good way to be able to filter for younger players and how they are performing. Its min and max values are the minimum and maximum ages in the player_data_2023 data frame, and the initial value is the same.

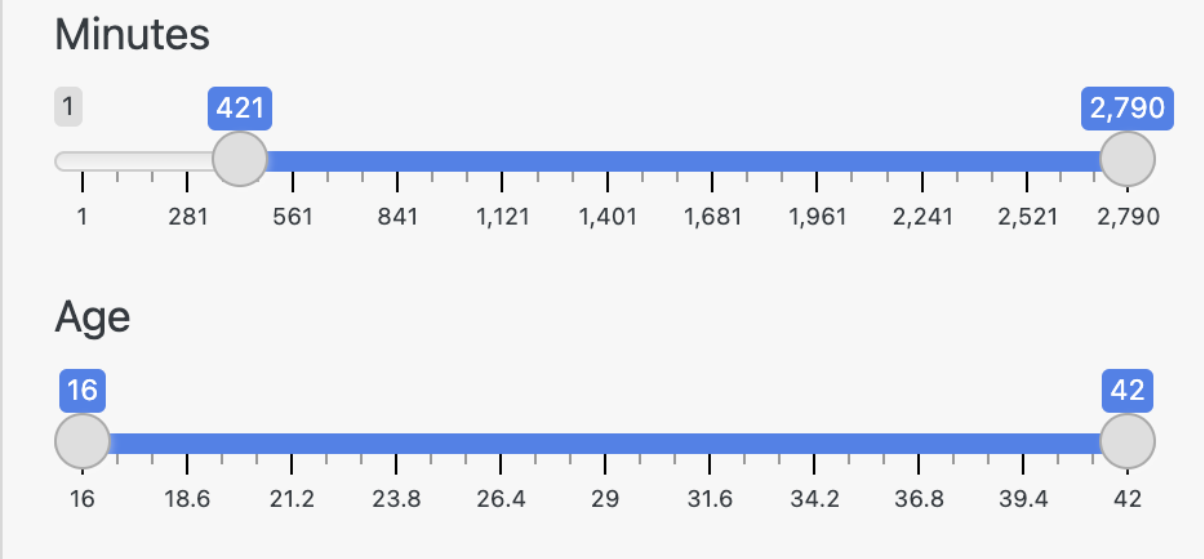


Figure 57 Slider inputs for Minutes and Age

5.5.1.6 Highlight Player in Plot

The final UI input added to the application was the highlight feature. This allowed the user to select a player from the data frame and highlight that players point in the scatter plot. The player is highlighted by making the point bigger and in a different colour. This was a useful for feature to be quickly able to identify a player in the plot.

```
63          selectizeInput(
64            "highlight",
65            "Search a player to hightlight: (in red)",
66            choices = NULL,
67            selected = "",
68            multiple = TRUE,
69            options = list(plugins = list('remove_button'))
70          ),
```

Figure 58 Highlight player select input

The difference with the highlight player input is the use of server side selectize. This means the input is dynamically updated in the server function. Like the other inputs it has an ID. It also allows for multiple selections. The difference is that the choices and selected argument are left as null. These will be updated in the server function which I will discuss later in this chapter.

5.5.1.7 Submit

The final input in the UI is the submit button. This is a simple button, but importantly like the other inputs contains a ID which is used in the server function to display the plots. When the submit button is pressed the ID is passed to the server and the server observes the button submit and displays the plot.

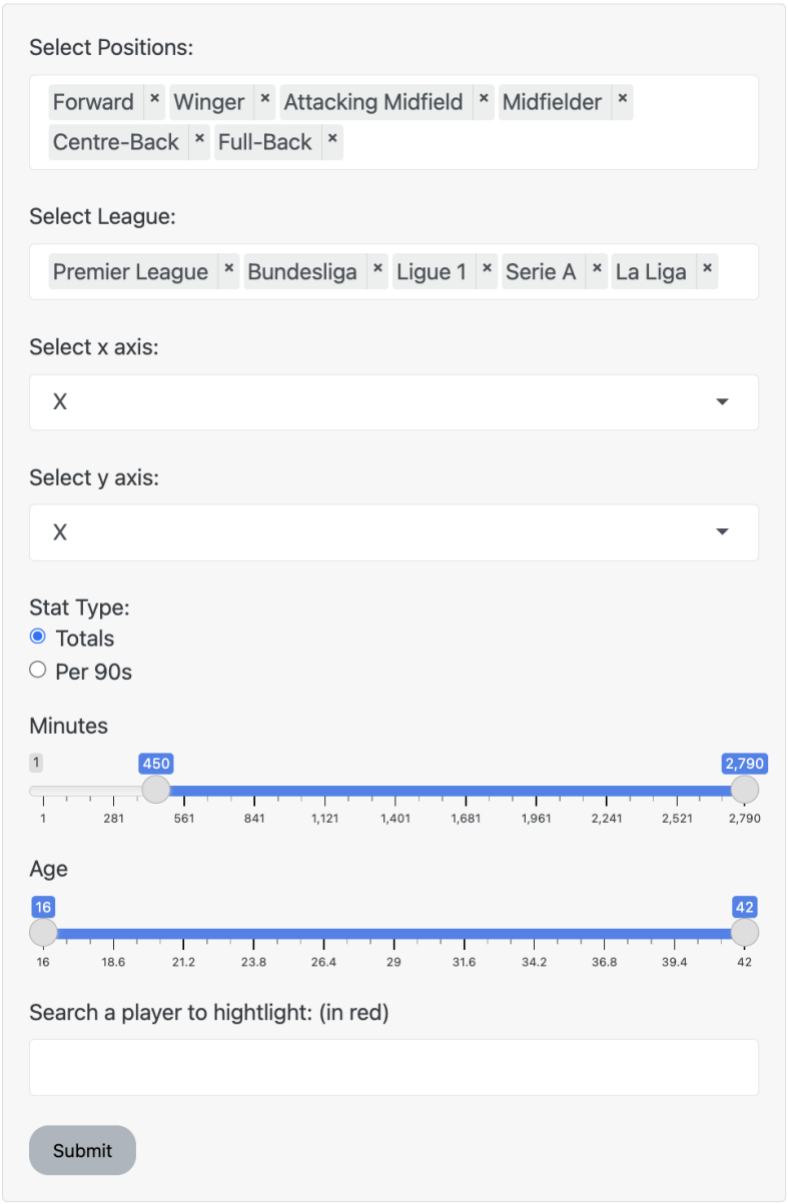


Figure 59 Player Scatter Plot Sidebar Layout with all Inputs

5.5.1.8 Historic Scatter Plot Inputs

The historic scatter plot page has a very similar UI function to the above. The only difference being that there is an extra select input for the Season. The historic data scatter plot displays player data from 2017/18 to the current season. The season input allows users to select which year to view in the plot.

```
83          selectizeInput(
84            "Season",
85            "Select Seasons:",
86            choices = unique(c(player_data_historic$Season_End_Year)),
87            selected = c(player_data_historic$Season_End_Year),
88            multiple = TRUE,
89            options = list(plugins = list('remove_button'))
90          ),
```

Figure 60 Season Select Input for historic stats page

This input works similar to the position and league input discussed above, with the choices being the values for Season_End_Year in the player_data_historic data frame. The selected argument assigns the default value in the input to all the season available. This input also uses selectize.js to provide a remove button.

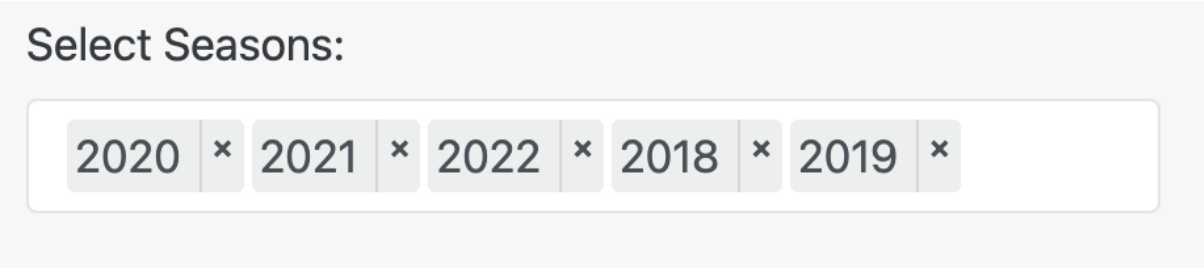


Figure 61 Season Select Input

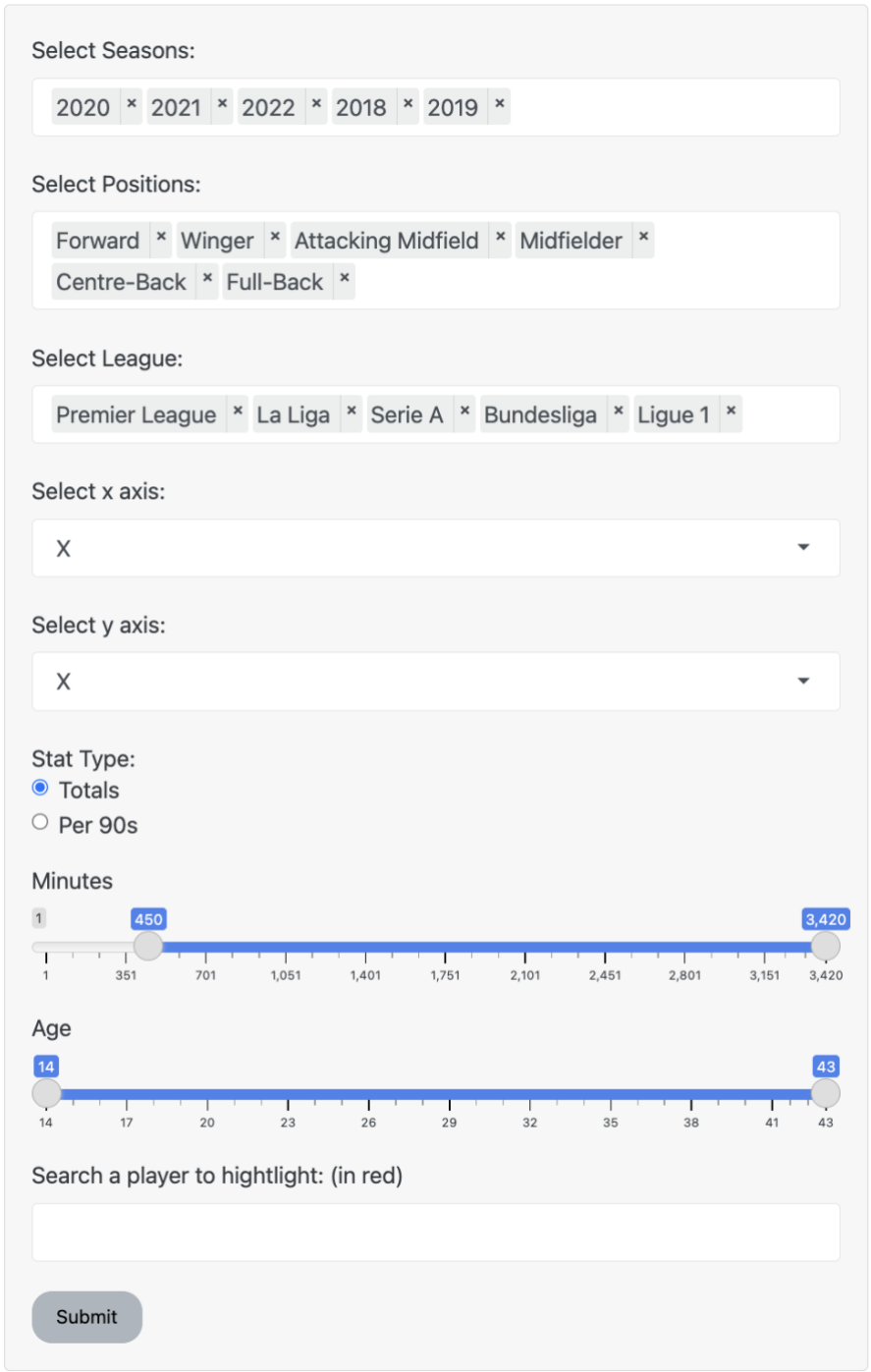


Figure 62 Historic Player Data Sidebar Layout with all Inputs

5.5.1.9 Reactive Player Data Function

The player data used in the scatter plots is a reactive data frame got from a reactive expression in the server function. The reactive function allows the data frame to be updated dynamically based on the input by the user. This is one of the places where the input IDs that were defined in the UI

function are used. The reactive data frame uses a series of IF Else statements to update the data frame. The reactive data frames outer layer is an if else statement that defines the stat type.

```
189  reactive_player_data <- reactive({
190    if (input$radio == "Totals") {
191      if (is.null(input$Comp) & is.null(input$Positions)) {
192        player_data_2023 |>
193          filter(
194            Min_Playing >= input$slider[1] &
195              Min_Playing <= input$slider[2] &
196              Age >= input$age[1] &
197              Age <= input$age[2]
198          )
199      } else if (is.null(input$Comp)) {
200        player_data_2023[player_data_2023$Position %in% input$Positions, ] |>
201          filter(
202            Min_Playing >= input$slider[1] &
203              Min_Playing <= input$slider[2] &
204              Age >= input$age[1] &
205              Age <= input$age[2]
206          )
207      } else if (is.null(input$Positions)) {
208        player_data_2023[player_data_2023$Comp %in% input$Comp, ] |>
209          filter(
210            Min_Playing >= input$slider[1] &
211              Min_Playing <= input$slider[2] &
212              Age >= input$age[1] &
213              Age <= input$age[2]
214          )
215      } else {
216        player_data_2023[player_data_2023$Comp %in% input$Comp &
217                           player_data_2023$Position %in% input$Positions, ] |>
218          filter(
219            Min_Playing >= input$slider[1] &
220              Min_Playing <= input$slider[2] &
221              Age >= input$age[1] &
222              Age <= input$age[2]
223          )
224      }
225    } else {
```

Figure 63 reactive_player_data where stat type is "Total"

The above code is half the reactive data frame for the player data. In the server function the ID is passed from the UI inputs using 'input$' followed by the input ID. The outer If-Else statement checks for the input from the stat type radio button using the input ID "radio". The above code shows the code for when the input is equal to "Total". This uses the player_data_2023 data frame. Within this there is a number of other If-Else and Else-If statements. The first on line 191, in the above figure, checks the input of Comp and Positions. It checks of both inputs are NULL, meaning nothing is selected, and if it is NULL returns player_data_2023 without filtering it. The data frame is then filtered based on the Age and Minutes slider inputs. The minutes and age filtering is done the same for each If-Else statement.

The Else-If on line 199 then checks if only the Comp input is NULL, if it is NULL then it filters the data frame by the Positions input. The %in% operator is used to check the data frame for the input ID. So for example if Forwards is the value selected in the Positions input then it checks the data frame for all entries where position is equal to forward. The next Else-If statement on line 207 does the opposite. It checks if the Positions input is NULL and if it is it filters the data frame based on the input from the Comp input. Finally, the last Else-If in the above code is for when neither the Comp or Positions input are NULL. When neither are NULL is filters the data frame based on both inputs using the & (and) operator. As mentioned previously, the slider inputs are used to filter each time.

If the stat type radio button is not equal to “Total” the below code is run.

```
225     } else {
226       if (is.null(input$Comp) & is.null(input$Positions)) {
227         player_per90_2023 |>
228           filter(
229             Min_Playing >= input$slider[1] &
230               Min_Playing <= input$slider[2] &
231               Age >= input$age[1] &
232               Age <= input$age[2]
233           )
234       } else if (is.null(input$Comp)) {
235         player_per90_2023[player_per90_2023$Position %in% input$Positions, ] |>
236           filter(
237             Min_Playing >= input$slider[1] &
238               Min_Playing <= input$slider[2] &
239               Age >= input$age[1] &
240               Age <= input$age[2]
241           )
242       } else if (is.null(input$Positions)) {
243         player_per90_2023[player_per90_2023$Comp %in% input$Comp, ] |>
244           filter(
245             Min_Playing >= input$slider[1] &
246               Min_Playing <= input$slider[2] &
247               Age >= input$age[1] &
248               Age <= input$age[2]
249           )
250       } else {
251         player_per90_2023[player_per90_2023$Comp %in% input$Comp &
252                             player_per90_2023$Position %in% input$Positions, ] |>
253           filter(
254             Min_Playing >= input$slider[1] &
255               Min_Playing <= input$slider[2] &
256               Age >= input$age[1] &
257               Age <= input$age[2]
258           )
259       }
260     }
```

Figure 64 reactive_player_data when stat type is not Total (is per 90)

This part of the reactive function works the same as discussed above however the players_per_90 data frame is used instead.

As well as the above there is a second reactive function that is to update the data frame for the historic data scatter plot. This reactive function works the same as the above except the player_data_historic and player_per90_historic data frames are used.

Because these data frames are reactive expression when they are used later in the server function they must be followed by parenthesis, for example, reactive_player_data().

5.5.1.10 Highlight Player Reactive Function / Update Select Input

As mentioned previously in this chapter the highlight select input makes use of server side selectize. This means the data frame that is used in the input is defined and updated in the server function. This is done so that the input can be updated to be the reactive_player_data data frame. If this wasn't done the players available to highlight in the select input may not actually be on the scatter plot because it has been updated by the user. By using the same reactive data frame as the plot this is not an issue.

```
357  observe({
358    updateSelectizeInput(
359      session,
360      "highlight",
361      choices = unique(reactive_player_data()$Name),
362      selected = "",
363      server = TRUE
364    )
365  })
```

Figure 65 update selectize input for the highlight player

This uses the updateSelectizeInput to dynamically update the input in the UI with the reactive_player_data data frame. By having server equal True the input can be updated in the sever function. Apart from this it works the same as the inputs in the UI function.

This same code is used to update the highlight input in the historic data page. The only difference being the data frame used and the input ID being passed.

Having defined and updated the highlight select input, the input can now be used to create a reactive data frame that can be used in the plot to highlight a player.

```
highlight_player <- reactive({
  reactive_player_data()[reactive_player_data()$Name %in% input$highlight,]
})
```

Figure 66 reactive data frame to highlight player

This reactive function uses the input from the highlight input to create a reactive data frame called highlight_player. This uses the reactive data frame created previously and checks it for the Name got from the highlight input. This data frame will be used in the scatter plot to highlight a player or players.

This is also done for the historic data page in the same way with the reactive_historic_players data frame.

```
highlight_historic_player <- reactive({
    reactive_historic_players()[reactive_historic_players()$Name %in% input$highlight1,]
})
```

Figure 67 historic reactive data frame to highlight player

5.5.1.11 Data Table Reactive Function

The final reactive functions used for the scatter plot pages is for the data tables. Below each plot on the scatter plot pages there will be a data table that has the raw data for the scatter plot. The reactive function is used to dynamically update the data frame based on the input in the X and Y Axis inputs.

```
dataTablePlayers <- reactive({
  reactive_player_data() |>
    select(Name,
           Position,
           Comp,
           Squad,
           input$x,
           input$y)
})
```

Figure 68 reactive function for data table

This function takes in the reactive_player_data data frame which is updated dynamically in the reactive function discussed previously. This reactive function then selects the columns to be used in

the data table. It selects the position, league and team of the player. It also selects the inputs that have been selected by the user for the X and Y axis.

5.5.1.12 Scatter Plot Output

The scatter plot is created using ggplot2 and plotly. Ggplot allows for the development of nice data visualisation graphics. Plotly is used to make these graphs interactive for the user. The code for the plot is inside an observeEvent() function. This function takes in the input from the submit button and then renders the plot when the button is clicked.

```
401  observeEvent(input$submit, {
402    output$scatterPlotPlayers <- renderPlotly({
403      p <-
404        ggplot(reactive_player_data(),
405               aes_string(
406                 x = input$x,
407                 y = input$y,
408                 text = "Name"
409               )) +
410        geom_point() +
411        geom_point(data = highlight_player(),
412                   color = "#ff6d00",
413                   size = 5) +
414        ggtitle("Football Scatter Plot") +
415        xlab(input$x) +
416        ylab(input$y)
417
418      ggplotly(p, tooltip = c("text", "x", "y"))
419
420    })
421    output$dt1 = renderDataTable({
422      dataTablePlayers()
423    })
424  })
```

Figure 69 Player Scatter plot code

The ggplot() function initializes the ggplot. It takes in two arguments for this plot. The first is the data frame used in the plot, In this case, the reactive_player_data data frame which we created previously. Notice the parenthesis after because it is a reactive data frame. This also means that the plot will update dynamically if the inputs change after rendering the plot with submit. Ggplot also take in an aesthetic argument (aes_string). In this a number of variables are declared. For this plot the x and y variables are given the value of the x and y inputs from the UI. The text variable is assigned the value from the Name column. These variables are used to display information on each

point, because this is a plotly graph this info will be shown when hovering on a data point in the scatter plot.

After initializing the ggplot, the geom_point() function creates the points on the scatter plot. This function takes in no arguments because the default value takes the data from the ggplot function. Following this there is a second geom_point() function. This is to highlight the player. This geom_point() has a data argument which uses the highlight_player() reactive data frame which was created previously. Like with the ggplot data, because this is a reactive data frame the highlighted point will update dynamically when the input changes. This geom_point() function also has a colour and size argument. The colour argument defines the colour of the point and the size defines the size of the point. Because this point is used to highlight a player, the point is a orange colour and is a bigger point.

The final three functions in the ggplot code sets the title of the plot using ggtitle() function and the label of the x and y axis using xlab() and ylab() functions. These use the inputs from the x and y input to set the labels.

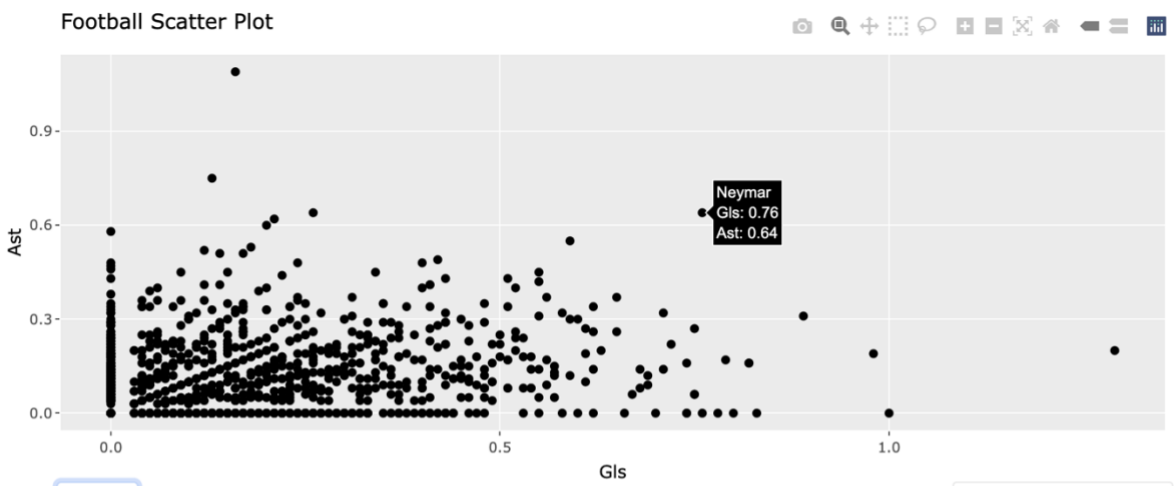
Having created the ggplot it is assigned the value of 'p'. This is so that it can be passed into the ggplotly() function. The ggplotly() function makes the plot interactive. The tooltip argument defines what is shown when hovering over a point in the plot. This is where the variables we defined in the ggplot function are used.

Finally, below the ggplot code is the data table. This takes in the data frame to be used in the table. In this case the reactive data frame that was created previously.

Having created the scatter plot and the data table, the output is then sent to the UI function using output$ followed by the ID of the plot. This output is then used in the UI main panel to create the plot and table.

```
75        mainPanel(verticalLayout(
76          plotlyOutput("scatterPlotPlayers"), DTOutput('dt1')
77        )),
```

Figure 70 Output from the server function used in UI function to show Plot and Table



Show 10 entries Search:

	Name	Position	Comp	Squad	Gls	Ast
1	Aaron Cresswell	Full-Back	Premier League	West Ham	0	0
2	Aaron Hickey	Full-Back	Premier League	Brentford	0	0
3	Aarón Martín	Full-Back	Bundesliga	Mainz 05	0.18	0.18
4	Aaron Ramsey	Midfielder	Ligue 1	Nice	0.06	0.06
5	Aaron Wan-Bissaka	Full-Back	Premier League	Manchester Utd	0	0

Figure 71 Scatter Plot and Data table

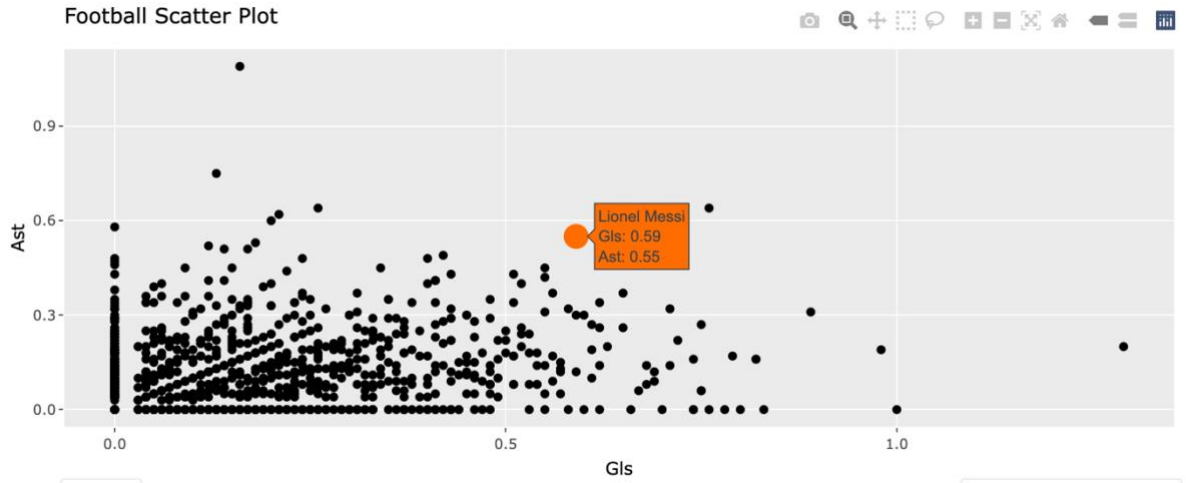


Figure 72 Scatter Plot using highlight feature

The Scatter plot and data table for the historic data page is done very similarly to the above. The only difference being the data frame used. One slight difference is that the variables defined in the ggplot() function contains a Season variable. This is so that the Season of each point can be included on the tooltip.

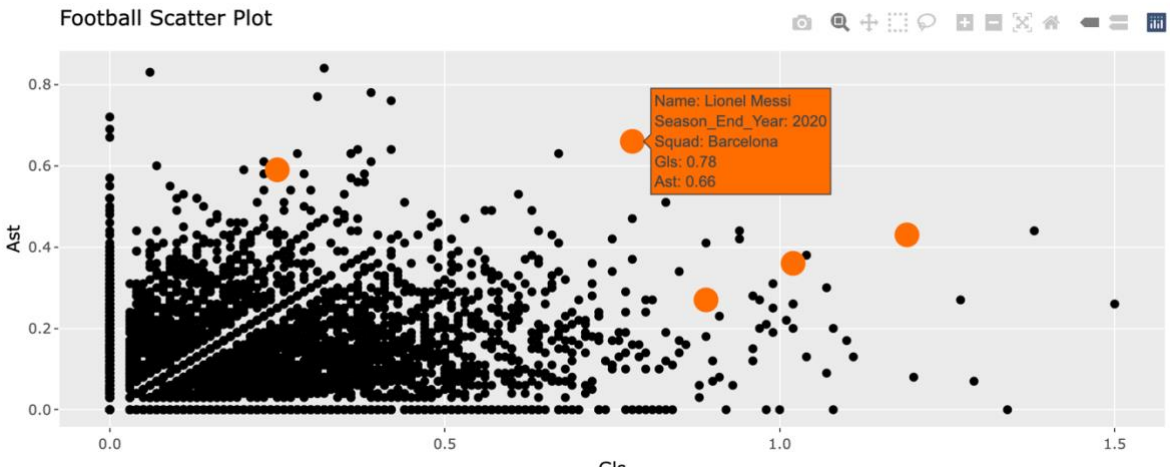


Figure 73 historic scatter plot with highlight feature

5.5.1.13 Saving Data Table

The scatter plots have a built-in functionality to save the plot using the plotly package. However is was also required that the user would be able to save the raw data that is displayed in the plot. This is done using the downloadButton() function. This button is added to the main panel where the plot and data table is outputted.

```
78          conditionalPanel(
79            condition = "output.scatterPlotPlayers",
80            downloadButton("saveData", "Download Data")
81          ),
```

Figure 74 download button for scatter plot data table

The download button is put inside a conditionalPanel() function which checks if the scatter plot has been rendered. This button has an ID “saveData” which is used in the server function.

Inside the server function, the downloadHandler() function is used to save the data.

```
449     output$saveData <- downloadHandler(
450       filename = function(){
451         paste('Player Data','.csv',sep='')
452       },
453       content = function(file){
454         write.csv(dataTablePlayers(), file)
455       }
456     )
```

Figure 75 code to save scatter plot data

This downloadHandler() function takes in a filename argument which defines the name of the file that is saved. The content argument is then used to save the data as a csv file using the write.csv() function. This takes in the data frame from the data table as well as the file path.

5.5.2 Player Pizza Charts

The idea for the player pizza chart feature in this application was to be able to nicely visualise how well a player performs in a variety of different statistics vs other players in the same position. This is done by ranking each player in each statistic. Each "slice represents a statistic. The size of each "slice" in the chart then represents how well a player performs in that statistic. The bigger the "slice" the better that player has performed.

5.5.2.1 Pizza Chart UI Inputs

Like for the scatter plots the pizza chart page contains a couple of select inputs. The main two inputs are the positions input and the player input. The position input is where the user decides which position to filter, and then the player input contains all the players from that position in the data frame for the user to select.

```
152                 selectizeInput(
153                   "pizzaPosition",
154                   "Select Position:",
155                   choices = NULL,
156                   selected = "",
157                 ),
158                 selectizeInput(
159                   "player",
160                   "Search Player:",
161                   choices = NULL,
162                   selected = "",
163                 ),
```

Figure 76 Pizza Chart select input

Like some of the inputs mentioned in the scatter plot section, these select inputs use server side selectize. Meaning the inputs are updated in the server function.

```
488   selected_player <- scouting_reports_2023
489
490   updateSelectizeInput(session,
491                        "pizzaPosition",
492                        choices = unique(selected_player$Position),
493                        selected = "",
494                        server = TRUE)
495
496   selected_player_position <- reactive({
497     selected_player[selected_player$Position %in% input$pizzaPosition,]
498   })
```

Figure 77 pizzaPosition updateSelectizeInput

This updateSelectizeInput updates the input in the UI. This input is then used to create a reactive data frame that filters based on the input of the pizzaPosition input. This reactive data frame is called selected_player_position. This data frame is then used to dynamically update the player select input so that only players in the selected position are available in the select input.

The reason for having the position input is for the comparison feature. The pizza chart page features the ability to compare two players. By having the user select a position, that position can then be used to only display players in the same position to compare.

```
164                 checkboxInput(
165                   inputId = "multi_select",
166                   label = "Compare Players",
167                   value = FALSE
168                 ),
169                 conditionalPanel(
170                   condition = "input.multi_select == true",
171                   selectizeInput(
172                     "player_comparison",
173                     "Search Player to compare:",
174                     choices = NULL,
175                     selected = ""
176                   ),
```

Figure 78 player comparison input

The comparison select input features a checkbox input that is used to allow the user to decide whether to compare players or not. If the check box is selected then the compare player input appears. This uses the "condition" expression to check the value of the multi_select input.

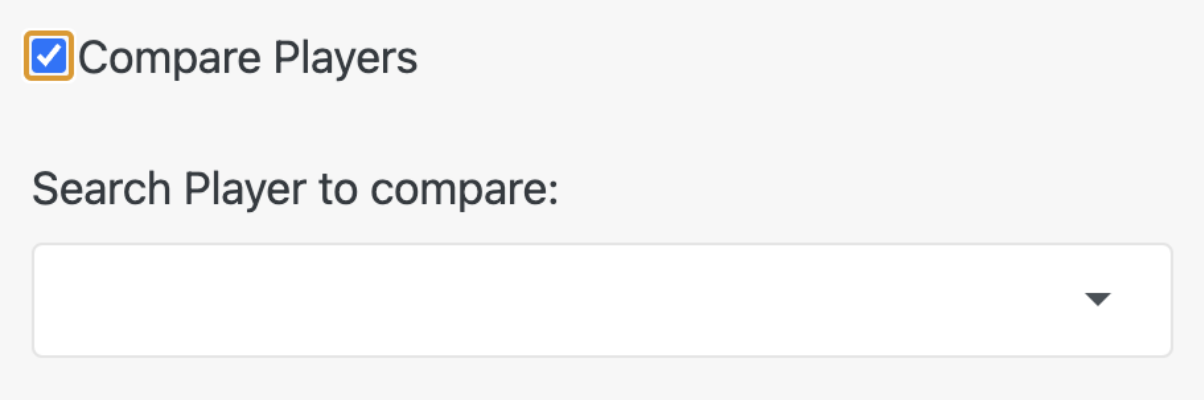


Figure 79 compare player UI

This compare players input is also updated in the server with the selected_player_position reactive data frame. so the players in the comparison input are always the same position as in the player input. This is done because players should only be able to be compared with players in the same position. There is no point in being able to compare a Centre-Back with a Forward because there data would be very different.

5.5.2.2 Indexing Data Frame for Pizza Chart

The data frame that will be used in the pizza chart is created in the observeEvent function of the pizza chart. This takes the input from the submit button and displays the chart when it is clicked.

```
514  observeEvent(input$submit1, {
515    selected_player <- reactive({
516      selected_player_position() |>
517        filter(Name == input$player)
518    })
519
```

Figure 80 selected_player data frame

This reactive data frame is created from the selected_player_position data frame which only contains the players in the selected position. It then filters this data frame for the name of the player from the player select input.

This reactive data frame is then subset to get the statistics that are required for the pizza chart. The pizza chart does not display every statistic, instead subset is used to select the 15 statistics to be shown in the pizza chart.

```
557  selected_player <- selected_player()[c(4, 5, 7, 10, 15, 16, 21, 24, 29, 32, 34, 36, 41, 46, 49), ]
558  selected_player$index <- 1:15
559  selected_player <- selected_player |>
560    mutate(
561      type = case_when(
562        index %in% 1:4 ~ "Attacking",
563        index %in% 5:8 ~ "Possession",
564        index %in% 9:12 ~ "Passing",
565        index %in% 13:15 ~ "Misc"
566      )
567    )
568  selected_player$type <-
569    factor(selected_player$type,
570           levels = c("Attacking", "Possession", "Passing", "Misc"))
```

Figure 81 getting statistics to be used in the pizza chart

In the above code the selected_player data frame is created from the selected_player() reactive data frame. It only includes the required statistics which are got at line 557 in the above. This data frame is then given a index column which index the data frame from 1 to 15. Another column is then created called type which assigns each statistic a type, Attacking, Possession and Defending. These types are used in the chart to colour each type a different colour.

This same process is then done for the compare_player data frame which is used to show the data for the player being compared.

```
    compare_player <- scouting_reports_2023 |>
      filter(Name == input$player_comparison)

    compare_player <-
      compare_player[c(4, 5, 7, 13, 21, 18, 29, 32, 41, 43, 48, 49), ]
    compare_player$index <- 1:12
    compare_player <- compare_player |>
      mutate(
        type = case_when(
          index %in% 1:4 ~ "Attacking",
          index %in% 5:8 ~ "Possession",
          index %in% 9:12 ~ "Defending"
        )
      )
    compare_player$type <-
      factor(compare_player$type,
             levels = c("Attacking", "Possession", "Defending"))
```

Figure 82 getting statistics for comparison player

5.5.2.3 Pizza Chart Position Templates

The method discussed above subsets the selected_player() reactive data frame to provide the statistics required for the pizza charts. However, the statistics that are displayed on the chart should not be the same for all positions. For example, the statistics that should be used for a forward should not be the same as for a defender, the forwards chart should display more attacking statistics while a defender's chart should display more defensive stats.

To achieve this, a series of If-Else statements are used to provide the templates for each position.

```
555
556     if (selected_player()$Position[1] == "Forward") {
557       selected_player <- selected_player()[c(4, 5, 7, 10, 15, 16, 21, 24, 29, 32, 34, 36, 41, 46, 49), ]
558       selected_player$index <- 1:15
559       selected_player <- selected_player |>
560         mutate(
561           type = case_when(
562             index %in% 1:4 ~ "Attacking",
563             index %in% 5:8 ~ "Possession",
564             index %in% 9:12 ~ "Passing",
565             index %in% 13:15 ~ "Misc"
566           )
567         )
568       selected_player$type <-
569         factor(selected_player$type,
570                levels = c("Attacking", "Possession", "Passing", "Misc"))
571     } else if (selected_player()$Position[1] == "Midfielder") {
572       selected_player <- selected_player()[c(7, 8, 15, 18, 20, 21, 23, 29, 32, 33, 26, 38, 40, 48, 44), ]
573       selected_player$index <- 1:15
574       selected_player <- selected_player |>
575         mutate(
576           type = case_when(
577             index %in% 1:3 ~ "Attacking",
578             index %in% 4:7 ~ "Possession",
579             index %in% 8:11 ~ "Passing",
580             index %in% 12:15 ~ "Defending"
581           )
582         )
583       selected_player$type <-
584         factor(selected_player$type,
585                levels = c("Attacking", "Possession", "Passing", "Defending"))
```

Figure 83 creating position templates for pizza charts

As the code above shows, the code to subset the selected_player() data frame to get the required statistics is put inside of an If-Else statement. This statement checks the value in the position column of the selected_player() data frame. The statistics used are then changed based on the position.

This same method is then used on the compare_player() data frame so that the same templates are used for the comparison charts.

5.5.2.4 Creating Pizza Chart

The code to create the pizza chart is wrapped in an If-Else statement that checks if the multi_select input and the player_comparison input are selected. If the multi select and player_comparison are selected the comparison chart is displayed, otherwise the normal chart is displayed.

```
555     if (input$multi_select && !is.null(input$player_comparison)) {
```

Figure 84 If-Else statement for displaying pizza chart

The code to create the single player chart can be seen below.

```
624      output$radarChart <- renderPlot({
625        color1 <- "#008a71"
626        color2 <- "#0362cc"
627        color3 <- "#ffa602"
628        ggplot(data = selected_player,
629               aes(
630                 x = reorder(Statistic, index),
631                 y = percentile,
632                 label = percentile,
633                 fill = type
634               )) +
635          geom_bar(data = selected_player,
636                   width = 1,
637                   stat = "identity") +
638          scale_y_continuous(limits = c(0, 100)) +
639          coord_curvedpolar() +
640          geom_hline(yintercept = seq(0, 100, by = 100),
641                     linewidth = 1) +
642          geom_vline(xintercept = seq(.5, 12, by = 1),
643                     linewidth = .5) +
644          geom_label(
645            color = "gray20",
646            fill = "oldlace",
647            size = 2.5,
648            fontface = "bold",
649            show.legend = FALSE
650          ) +
651          scale_fill_manual(values = c(color1, color2, color3)) +
```

Figure 85 creating single player pizza chart

The colours defined at the beginning are the colours used to distinguish the stat types in the chart. The pizza chart is then created by first creating a regular ggplot bar chart. This ggplot function takes in the selected_player data frame as an argument. In the aesthetics argument the X axis is defined as the Statistics which are sorted by the index we added earlier. The Y axis is the percentile value which controls how big the “slice” will be. The better percentile rank the bigger the “slice”. The label variable in aes() displays a label on each slice and the value is set to the percentile, so the user can see what percentile the player is in each statistic.

Having created the normal bar chart, the coord_curvedpolar() function is used to create a round chart. Also, the scale_y_continous() function is used to define the sale of the chart. So a player with percentile 100 is at the very edge of the chart.

The geom_vline and geom_hline functions are used to add lines between the slices and around the chart.

The resulting pizza chart can be seen below.

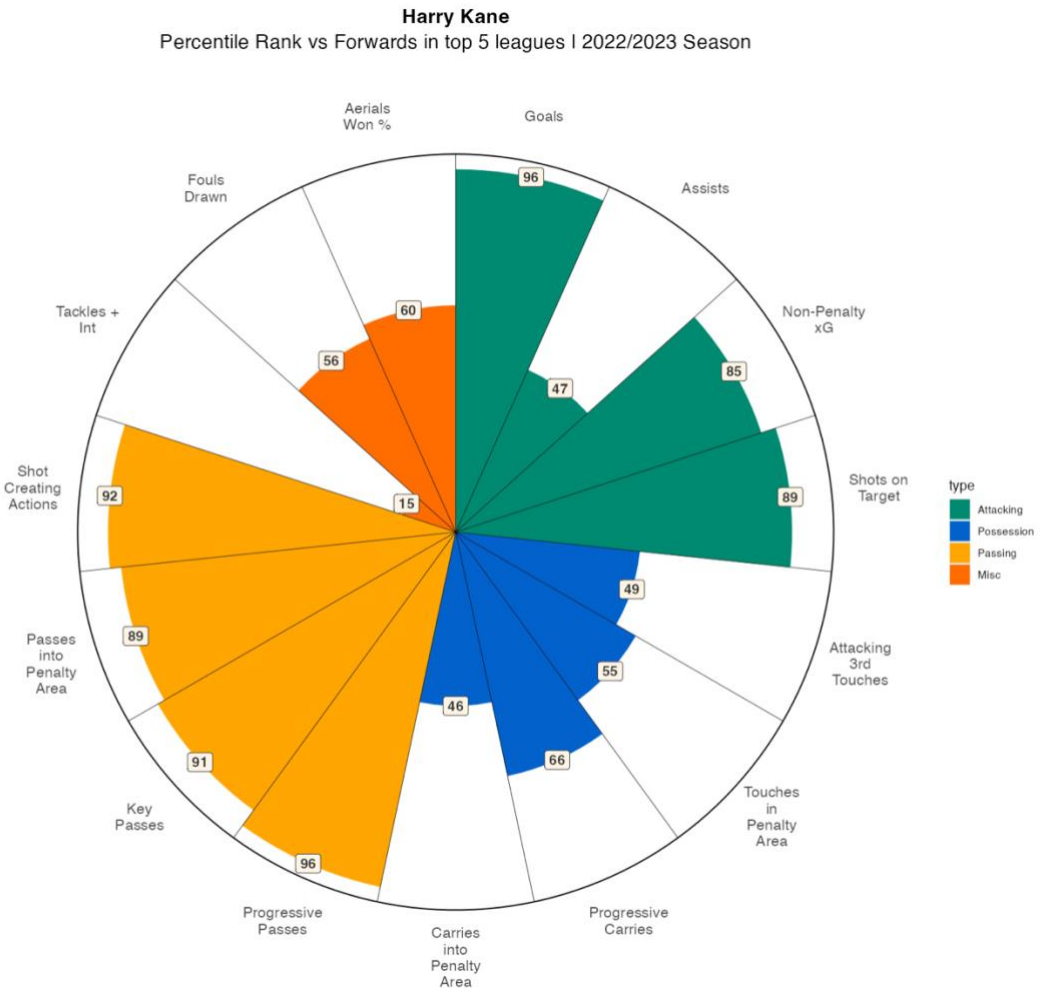


Figure 86 single player pizza chart

If the comparison player is selected the other chart in the If-Else statement is used. This code is the same with one exception.

```
556       output$radarChart <- renderPlot({
557         color1 <- "#008a71"
558         color2 <- "#0362cc"
559         color3 <- "#ffa602"
560         ggplot(data = selected_player,
561                aes(
562                  x = reorder(Statistic, index),
563                  y = percentile,
564                  label = percentile,
565                  fill = type
566                )) +
567           geom_bar(data = selected_player,
568                    stat = "identity",
569                    width = 1,) +
570           geom_bar(
571             data = compare_player,
572             stat = "identity",
573             width = 1,
574             alpha = 0,
575             color = "black",
576             size = 2
577           ) +
578           scale_y_continuous(limits = c(0, 100)) +
579           coord_curvedpolar() +
580           geom_hline(yintercept = seq(0, 100, by = 100),
581                      linewidth = 1) +
582           geom_vline(xintercept = seq(.5, 12, by = 1),
583                      linewidth = .5) +
584
585           geom_label(
586             color = "gray20",
587             fill = "oldlace",
588             size = 2.5,
589             fontface = "bold",
590             show.legend = FALSE
591           ) +
592           scale_fill_manual(values = c(color1, color2, color3)) +
```

Figure 87 code for comparison pizza chart

As you can see in the code above, on lines 570 to 576 an extra geom_bar is added. This is to display the data from the compare_player data frame. This creates a black outline around each slice which is also slightly thicker than the normal outline to shows the player being compared.

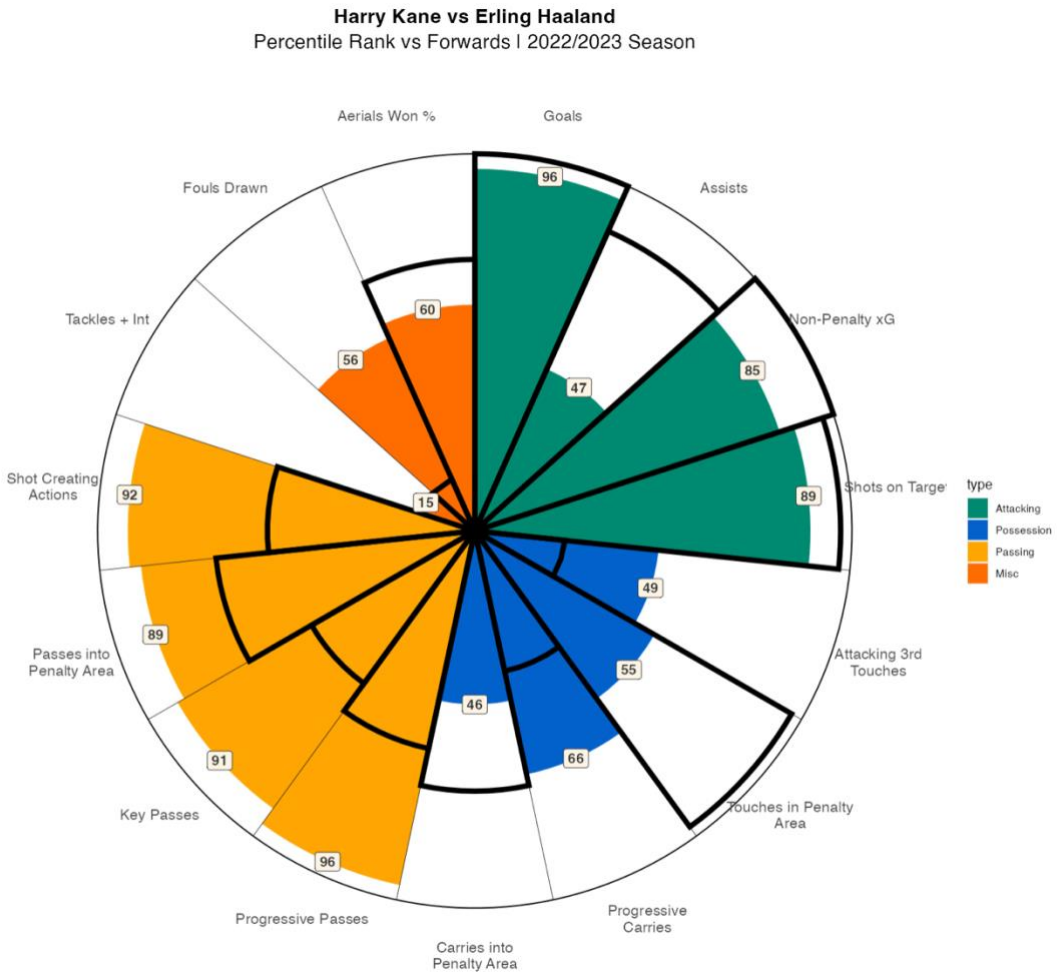


Figure 88 player comparison pizza chart

5.5.2.5 Displaying Output in UI

Like with the scatter plots, the ID of the output is passed from the server to the main panel in the UI to be displayed.

```
180                 mainPanel((plotOutput(
181                   "radarChart", height = "80vh"
182                 )))
```

Figure 89 output pizza chart in UI

This plotOutput() function takes in the plot ID and sets the height to 80vh. This means the chart changes based on the size of the screen.

5.5.2.6 Saving Pizza Chart

To allow the user to download the pizza chart the downloadButton() function is used in the UI function.

```
205             conditionalPanel(
206               condition = "output.pizzaChart",
207               downloadButton("save", "Download Plot")
208             )
```

Figure 90 save pizza chart button

This button is put inside a conditionalPanel() function. This function checks if the chart has been rendered in the UI, if it has the button is then displayed to the user along with the chart. The downloadButton() function takes in two arguments. The first is the ID of the button, this will be used in the server function to save the chart. The second is the buttons label.

```
367         output$save <- downloadHandler(
368           filename = function(){
369             paste(selected_player$Name[1],'.jpeg',sep='')
370             },
371           content = function(file){
372             ggsave(file,plot=pizzaChart, scale = 1.7)
373           }
374         )
```

Figure 91 code to save pizza chart

In the server function of the application the downloadHandler() function is used to save the chart. This function takes in two arguments. The first is the filename argument. This defines the name of the file, as well as the file type. In this case the file is saved as the selected players name as a jpeg file. The second argument is the content argument. This content function takes in the file argument which is the file path where the content will be saved. This uses the ggsave() function to save the plot. This function takes in the file path as well as the plot that is saved.

This same method is used in the comparison chart to save it. With the exception being the name of the file being saved and the plot used in the ggsave() function.

```
793         output$save <- downloadHandler(
794           filename = function(){
795             paste(selected_player$Name[1], " vs ", compare_player$Name[1], '.jpeg',sep='')
796             },
797           content = function(file){
798             ggsave(file,plot=comparisonChart, scale = 1.7)
799           }
800         )
```

Figure 92 saving comparison chart

5.6 Conclusion

This chapter outlines the implementation of the application. The implementation process was done using the Scrum methodology. This involved breaking the process down into sprints, each sprint had the goal of completing a certain feature or functionality. Any goal not completed during a sprint is passed onto the next sprint.

In the chapter I discuss the development process of the application. This is done by talking through the code and what was done to implement the different features and functionality of the application. This section contains a number of code snippets which give an in depth look at the code. The implementation is broken down into two sections. The first section discusses the data scraping and processing that was undertaken. The second section of the implementation chapter outlines the coding pf the applications features and functionality.

Overall the implementation process was successful. All the functionalities and features that were outlined in the requirements and design chapters were implemented without any major bugs or issues.

6 Testing

6.1 Introduction

The testing chapter will be used to discuss the testing that was performed on the application. The testing of the application is split into two section. Firstly, functional testing was carries out on the application. Functional testing tests the application against the functional requirements that were gathered during the requirements process. The functional testing is concerned with whether the application works as it should, not whether it is easy and enjoyable to use. The User testing is done by giving users access to the application and getting feedback from them about their experience. It focuses on whether the application is easy and enjoyable to use.

The testing process is a vital step in the software development process because it can help discover bugs, identify improvements that can be made and whether the application is appealing to users.

6.2 Functional Testing

The functional testing was used to see whether the functionality of the application performed as was expected. This was done using the black box testing technique. This testing technique does not focus on the internal logic of the application, rather it only looks at whether the expected output is the same as the actual output.

The functional testing is divided into three sections: Navigation, Inputs, and Plot Outputs.

6.2.1 Navigation

Test No	Description of test case	Input	Expected Output	Actual Output	Comment
1	Launching Shiny Application	User goes to the apps URL.	User navigates to the Shiny app	Users is brought to the Shiny App.	When application has been idle load time can be roughly 5 to 10 seconds
2	User is navigated to Player Scatter Plot page	User clicks ‘Player Scatter Plots’ tab in navbar	User is brought to scatter plot page	User is brought to scatter plot page	User starts on this page when launching application.
3	User is navigated to Historic Player Scatter plots	User clicks the ‘Historic Player	User is brought to historic	User is brought to historic player	

		Scatter plot' tab in navbar	player scatter plot page	scatter plot page	
4	User is navigated to Pizza Chart Page	User clicks 'Player Pizza Chart' tab	User is brought to the player pizza chart page	User is brought to the player pizza chart page	
5	Page is refreshed	User refreshes the page	Page refreshes the page to clear inputs and plots, but stays on the same page	Inputs and plots are cleared when page is refreshes but user is navigated back to the player scatter plot page	

6.2.2 Inputs

Test No	Description of test case	Input	Expected Output	Actual Output	Comment
1	All positions are selected by default in the positions filter.	Navigate to Player Scatter Plot page	The position input by default has all positions selected	All positions are selected in the input	
2	Unselecting position from input	User clicks remove icon on position	Position is unselected from position input	Position is unselected	
3	Selecting positions in input	User selects position from select input dropdown	Position becomes selected in the input	Position is selected	
4	All leagues are selected by default in leagues input	Navigate to player	All leagues are selected by default	All leagues are selected	

		scatter plot page			
5	Unselecting league from input	User clicks remove icon on league	League is unselected	League is unselected from input	
6	Selecting league from input	Users selects a league from the select input	League becomes selected	League is selected	
7	Selecting Stat for x axis	User selects a stat from the x axis input	Stat is selected	Stat is selected	
8	Selecting stat for y axis	User selects stat from the y axis input	Stat is selected	Stat is selected	
9	Selecting player to highlight in plot	User selects a player from the dropdown in the highlight select input	Player is selected to be highlighted	Player is selected	Can select multiple players
10	Removing player from highlight input	User clicks remove icon	Player is unselected from the highlight input	Player is unselected	
11	User can save data from the data table	User clicks save data button	Data is saved as csv	Data is saved as csv file	
12	On Pizza Chart page, selecting position	User selects a position	Position is selected	Position is selected	
13	Selecting player for pizza chart	User selects player for the pizza chart	User can only select player with same position as selected position	Only players with same position available	If no position selected there is no players in player input

14	Compare player checkbox displays compare player input	User selects the compare player checkbox	The compare player input is displayed	The compare player input is displayed when checkbox is ticked	
15	No position selected and compare is selected	User clicks compare player checkbox but doesn't select a position	No players in compare player select input	No players in compare player select input	
16	User can save pizza chart	User clicks save chart button	Chart is saved	Chart is saved	

6.2.3 Plot Outputs

Test No	Description of test case	Input	Expected Output	Actual Output	Comment
1	Displaying the scatter plot	User clicks the submit button	Scatter plot is displayed, along with data table	Scatter plot is displayed as well as the data table	
2	Position and league filters	Users selects the preferred leagues and positions to be displayed	Only players from the selected leagues and positions are on the scatter plot	Scatter plot is filtered based on user inputs	
3	X axis is what the user selected in the input	User selects a stat in the x axis input	Selected stat is displayed on x axis	Selected stat is displayed on the x axis	
4	Y axis is what the user selected in the input	User selects a stat in the y axis input	Selected stat is displayed on y axis	Selected stat is displayed on the y axis	

5	Data table displays player info and x and y axis stats displayed	User selects the x and y axis inputs	Data table displays with columns of player info like name, team, position, and the x and y axis stat	Data table has player info and x and y axis stats	Data table only shows players in the scatter plot
6	Save data button is displayed	Users clicks submit button to render outputs	Save data button is displayed along with the plot and table	Save data button is displayed	
7	Data frame changes based on stat type checkbox	User selects either Total or Per 90 in the stat type input	If Total is selected the totals data are displayed in scatter plot, if per 90 is selected the per 90 data is displayed	Totals data is displayed when total is selected, per 90 data is displayed when per 90 is selected	
8	Minutes slider filters data in scatter plot	User selects min and max minutes value using slider input	Players whose minutes are in between min and max value of slider are filtered	Players whose minutes are in between min and max value of slider are displayed in plot and table	
9	Age slider filters data in scatter plot	User selects min and max age value in age slider	Only players in between them ages are in scatter plot and data table	Only players in between min and max age are displayed	
10	Highlighting player in scatter plot	User selects a player or players in the highlight input	Players point or points in the scatter plot is displayed in orange	Players point or points are displayed in orange in scatter plot	
11	Unselecting highlighted player	Clicks remove icon on	Orange highlight point is	Orange highlight point is	

		selected highlight player	removed from scatter plot	removed from plot	
12	Displaying all players years in historic scatter plot	User selects all the season and all player season are shown	All player season are shown	All player seasons are shown	
13	Unselecting seasons from scatter plot	User clicks remove icon on a season in the season input	All player point for that season are removed from plot	All player points for that season are removed	
14	After submit is clicked plot will update when inputs change	After the submit button has been clicked, the user changes the inputs	Scatter plot and data table are dynamically updated as the inputs change	Plot and table updates dynamically	
15	Displaying pizza chart	Users selects the position and then a player and clicks submit	Player pizza chart is displayed	Player pizza chart is displayed	
16	No Player selected	User selects no player and clicks submit	Chart is not displayed	Chart is not displayed	Error: 'gpar' element 'fontsize' must not be length 0
17	Displaying compare player chart	User selects a position and player, checks compare player and selects a player to compare	Pizza chart is displayed with black outline showing the comparison player	Players comparison chart is displayed	
18	No comparison player selected	User selects a position	Pizza chart is not displayed	Pizza chart is displayed but without	

		and player, and clicks compare player checkbox but doesn't select a comparison player		any comparison	
19	Save pizza chart button is displayed	User selects player and submits to render chart	Save chart button is displayed along with pizza chart	Save button is displayed	

6.2.4 Discussion of Functional Testing Results

Overall the results of the functional testing carries out on the application were positive. Most of the functionality worked as expected with only a few minor issues identifies that may need to be corrected. However none of these minor issues affected the functionality of the application.

Without any complex navigation functionality in the application it was expected that the app would pass all the functional testing on navigation, which it did, however one thing to note from the navigation testing was that when the page was refreshed the user was navigated back to the player scatter plot page even if on one of the other pages. Effectively the user is redirected to the home page every time the application is refreshed. The other finding from the navigation testing was that when the application is launched it can take a few seconds to load if the application has been in an idle state. This is a result of the hosting platform used for hosting and deploying shiny apps putting apps in an idle state when they haven't been launched recently.

Similarly, the inputs functional testing was a success for the majority of it. All the inputs for the scatter plot works how expected with the exception of one thing. When the user selects no position or no league in the league and position inputs, there is a bug where the scatter plot displays the data as if all were selected. Aside from this bug the scatter plot inputs work as expected. The pizza chart pages inputs all work as expected.

Finally, the plot outputs functional testing were mostly successful and performing as expected. For the scatter plots the outputs all work as expected based on the inputs, with the exception of the minor bug relating to no league or position being selected as mentioned above. A couple of notes were made during the pizza chart output functional testing. When the user selects no player and clicks submit no chart is displayed as expected, however there is an error message displayed on the screen. This is something that can solved by adding some form validation to the player input to make the input required. One other thing to note in relation to the pizza charts. When using the compare player functionality if the user selects a player and then clicks the compare player checkbox but does not select a player to compare, instead of no chart being displayed the chart is displayed without any comparison.

Overall the functional testing on the application was successful with only a couple minor notes to take away from the testing. These are not major functional issues which can be amended and did not affect the overall functionality of the application.

6.3 User Testing

The functional testing is about testing whether the application works as expected. It is tested vs the functional requirements that were outlined in the requirements section of this project. The user testing is about testing whether the application is easy to use, and whether it is an enjoyable experience for the user.

6.3.1 Method

The user testing was performed by giving users the hosted version of the application. This allowed the users participating in the user testing to have full access to the application. Users were then asked to go through the different functionality of the application. Having used the application the users then filled out a post testing survey about the application. These questions and the corresponding answers help gain a understanding of how users view the application. It helps identify the positive features of the application, as well as the features that can be improved on. The user testing was conducted with 4 participants.

6.3.2 Results

The overall results of the user testing were positive. With 100% of users saying they would be likely to use the application again. The first specific question that the user were asked about was the navigation, how easy the website was to navigate around. The applications navigation is quite simple so the expectation was that it would be a positive response from users, which it was. All users that took part were able to easily navigate between the different pages.

The next part of the user testing was testing the scatter plot functions. Users were first asked about the filters and inputs for the plot. With the filtering of the scatter plots being a vital piece of functionality for the application it was important that it was easy and intuitive to use. 50% of the users had no trouble understanding how to use the inputs and filtering, and the other 50% found it slightly difficult to understand them. Although it was hoped that the users would unanimously find the inputs and filtering easy to use, no users found it difficult to use. Due to some users finding it slightly difficult to use it could be worth adding some helpers to the inputs, like a helper icon beside each input. The second task for users in relation to the scatter plot features was finding players on the charts. All users that took part said they were easily able to view players and find certain players they wanted in the scatter plot. It was important that finding players on the scatter plot is easy to use for users, so it is a positive that all users believed this. The plotly package to make interactive scatter plots and the highlight player functionality were important features for this.

Moving onto the player pizza chart functionality, users were asked if the chart was easy to understand and similarly if it was an easy way of understanding player performance. All users found the pizza chart easy to use, and importantly all users found it an effective way of judging player performance. This was important because the idea behind the player pizza chart feature was that it would be a useful way of easily understanding what a player is good at. Finally, on the pizza chart function, users all found the comparison feature an easy and useful way of comparing two players.

Having worked through the features and functionality of the application during the user testing process the users were then asked for the features of the application that they like most, as well as features they would have liked the application to have or where it could be improved. The most popular choice of positive features in the application was the pizza chart feature. The historical stats page was also positively regarded by the users for the ability to view player performance over a number of past seasons. Finally, the highlight player functionality was a positive for its ability to easily identify players in the scatter plot that may contain a large amount of data points.

On the features that could be improved or that could be added, a common response was for more information on players in the scatter plot, for example adding the ability to click the point in the scatter plot and be shown a player profile. This is a good feature that could be added to the scatter plots. Other features that users suggested were the ability to filter the pizza chart by leagues and the ability to highlight players in a different colour if selecting one then one to highlight.

Overall, the results of the user testing was a success with most users finding the application easy and enjoyable to use. Both the scatter plot and pizza chart features were positives. The users also provided some valuable feedback about where there could be improvements to the application. With 100% of users answering that they would be likely to use the app again it shows that the application was enjoyed by the users participating in the testing.

6.4 Conclusion

In conclusion, the testing section of this development process was an important for identifying whether the application works as expected with the functional testing. Followed by testing with user whether the application is easy and enjoyable to use. The functional testing did not discover any major bugs or issues with the application, and any minor issues discovered can be fixed. The user testing was important for understanding what others think of the application. The user testing was a success with users overall enjoying the features and functionality of the app. It also provided a few suggestions that can be added to further improve the users experience on the app. The fact that 100% of the users that took part in the testing said they would be likely to use the application again is a very positive piece of feedback.

7 Project Management

7.1 Introduction

The project management section will outline how the project was managed during the development process. In this chapter I will discuss the phases of the project, as well as the tools used for project management. The use of Scrum methodology, which was discusses in the implementation chapter, was very important for managing the projects development. This involved breaking the project down into sprints. There were 9 sprints in total, with each sprint lasting 2 weeks. During each sprint meetings were held with my supervisor to outline what will be done and what has been done during a sprint. GitHub was an important tool for project management, this will be discussed in this chapter.

7.2 Project Phases

The projects development was split up into a number of stages. By splitting up to process the management of the project was made easier. In this section I will discuss the stages in the development of this project.

7.2.1 Research

The first stage of the project was the research stage. This involved researching a topic to do with the application that was being developed. For this the research section discusses data visualisation and the techniques that can be used to visualise data, as well as the use of data visualisation in sport. As a passionate fan of sports, with an interest in data and how it can be used, this project was a particularly interesting area for me.

7.2.2 Requirements

The next stage of the process was the requirements section. This involved researching the application area to provide a clear picture of what the application should be. This was done in a number steps, including researching similar applications, surveying potential users, and then outlining the functional and non-functional requirements of the application.

7.2.3 Design

The design phase of the project was split into two sections: the program design and the user interface design. The program design outlines the design for the coding of the application. This

makes the coding stage easier. The UI design section outlines how the application will look. By doing the design phase before the implementation, it made the implementation process easier. Instead of implementing the project without any clear plan, the design phase outlines how the application should be implemented. This was an important process in the management of the project.

7.2.4 Implementation

The implementation phase was the coding and development of the application. The implementation phase was done using Scrum Methodology. This involved breaking down the phase into a series of sprints. Each sprint lasted 2 weeks with the goal of completing certain features or functionality. At the start of each sprint the goals and task of each sprint would be outlined, and any backlog from the last sprint is passed over. During each sprint I met with my supervisor to outline what has been completed in the sprint as well as what will be done. By breaking down the implementation into a number of sections it helped to better manage the development of the application.

7.2.5 Testing

The testing phase of this project involved testing the functionality of the application, as well as getting users feedback on the application during the user testing. The functional testing phase tested whether the application worked as it was expected to. While the user testing was done to understand whether the application is easy and intuitive to use. The feedback from the users was managed using google forms, which the users filled out and provided feedback on a series of questions and tasks.

7.3 Project Management Tools

7.3.1 GitHub

GitHub was a very important project management tool during the development of this application. It was used for version control of the code. The code is committed to the projects GitHub repository. This GitHub repository is connected with the project on my device, so every time the code changed locally the new code can be committed to GitHub. GitHub desktop was used during the development. This allowed me to quickly and easily commit new code to the GitHub repository when it changes locally.

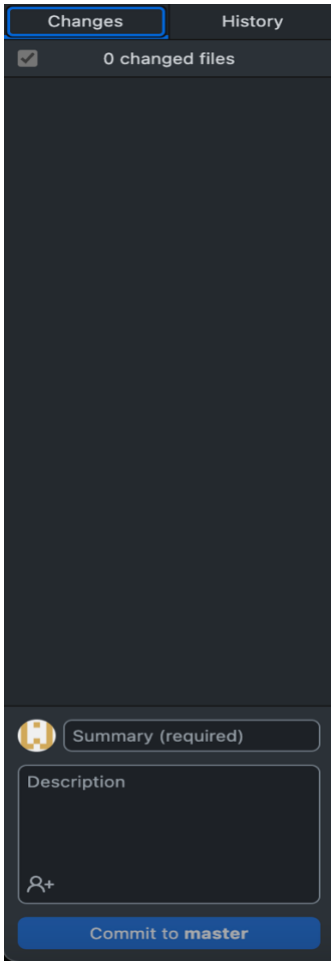


Figure 93 GitHub desktop interface

The above figure shows how GitHub desktop is used. When the files change locally they appear in the GitHub desktop interface and can be committed easily to GitHub by giving the commit a summary and description and then clicking commit.

As well as being used to easily commit new code to GitHub, GitHub desktop is used to easily keep track of changes to the code throughout the development process. The history tab has all the previous commits available, this can be important for fixing bugs that may appear by being able to easily return to a previous commit.

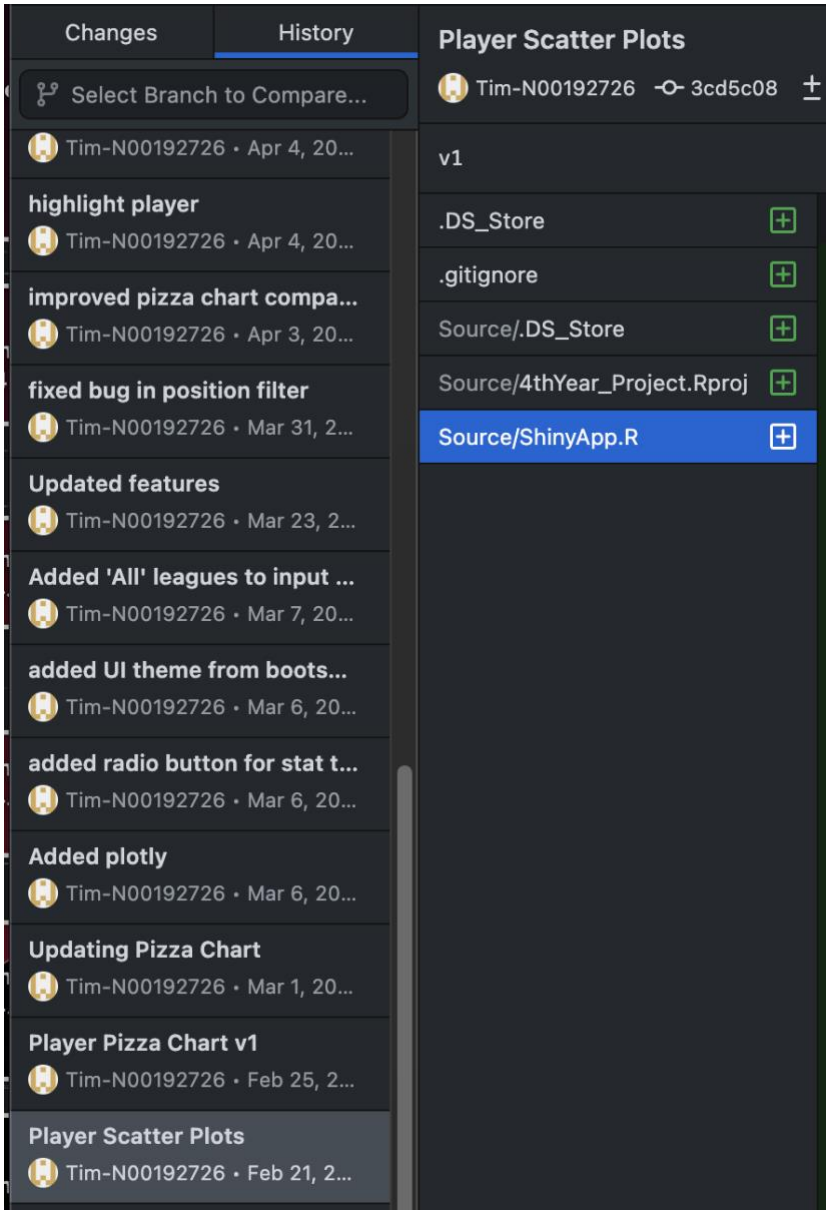


Figure 94 Commit history in GitHub desktop

7.4 Conclusion

Overall, the management of the project was a vital process during the development phase. It made the implementation of the application easier by breaking down the features and functionality into a number of sprints. Also, meeting with my supervisor each week was important as it allowed me to outline what has been done as well as plans for what will be done. For these reasons the use of Scrum methodology was vital for the development of this project.

GitHub and GitHub desktop were important tools for the management of the project. This allowed me to easily commit new code to GitHub as well as view previous commits and return to previous commits for fixing bugs.

8 Conclusion

This chapter will summarize this report as well as the development process. It will begin with a reflection on the project and the development of a major software project. I will then discuss some future work that could be done on the application. Finally, I will end the chapter with some final thoughts on the project.

8.1 Reflection

8.1.1 Views on the project

As a passionate sports fan and someone who is very interested in how data is and can be used in sport, building an application that moulds these interests was a very enjoyable process. Having not had any experience with the R programming language and the Shiny web application framework I was excited to be a able to develop an application with these technologies. I believe the completed application has been a success and met my expectations for what the final product would be when I started the project. I have always found data visualisation tools, particularly those for sports data, to be very useful and informative and I believe this application can be a useful and enjoyable tool for users.

8.1.2 Completing a large software development project

This project has provided valuable experience in developing large software project. The development process was completed using Scrum methodology. This was important for managing the workload of the project, by breaking the project down into a series of sprints it stops the development process becoming overwhelming.

The experience of developing a large software project also showed the importance of taking the process one step at a time. By gathering the projects requirements and outlining the design of the application before starting the applications implementation makes the coding and development of the app easier. The testing was then an important step in completing the project by testing whether the app works as expected and then getting feedback from users on the application and allowing for this feedback to be implemented. This is an area I believe I could improve with future projects by conducting the testing at an earlier stage to allow more time to implement all possible improvements.

8.1.3 Working with a supervisor

A vital part of this projects development was working with a supervisor. Every week I would meet with my supervisor and discuss the progress made and what I was currently working on, as well as

what I would be working on in the future. By having these regular meetings it gave me an opportunity to receive essential feedback on my project and application as it was being developed.

8.1.4 Technical skills

The main technical skill developed during the project is the R programming language. Having had no experience with the R language I believe I have learned a lot and would feel comfortable in developing other projects using R. Another technical skill learned is in the Shiny technology. Shiny is a web application framework for R. Having used Shiny for building this application I would feel confident in developing other web applications using it.

8.1.5 Further competencies and skills

As well as the technical skills developed during this project, there was a number of other important skills learned and required. One of the main skills used has been project management. This was a vital skill for the development of this project because it stops the work from becoming overwhelming. The Scrum methodology was very important for this.

Another key skills needed during the development was problem solving. This is an important skill for any software development project. This was particularly necessary when processing the data to be used in the project. When the data was scraped originally it was not in a form that could be used in the application. Problem solving skills were important in processing the data into a form that suits the needs of the applications features.

8.2 Future Work

Although I am very happy with the applications development and the final product, there is always room to improve an application like this. The main way I believe this project could be improved in the future is the data available to it. In its current form the application only features data on Europe's top 5 leagues, and although these are the most popular leagues and feature the most popular players, there is a number of other leagues that could be added to the application to provide even more depth. By adding further leagues it would give user more players to compare with, and also give users the ability to identify players who are not playing in the top 5 leagues but are performing well.

Another future improvement that could be added to the project is the addition of predictive data analysis. R offers the compatibility to perform data analysis using different machine learning techniques. This would be a nice feature to integrate into the application to not only be able to view player performance using data visualisation but also predict player or team performance using machine learning techniques.

8.3 Final Thoughts

Overall, I am very happy with the application has been developed and have enjoyed the development process. The application has met the expectations I had when beginning the project and have met the requirements and design I outlined in those chapters. The results of the user testing were positive with all users saying they would be likely to use the application. However, there is also room to improve the application in the future. This project has also thought me a number of new skills like the R programming language and R Shiny and has given me valuable experience in developing major software projects which will be important in the future.

9 References

Ajibade, S. S., & Adediran, A. (2016). An overview of big data visualization techniques in data mining. International Journal of Computer Science and Information Technology Research, 4(3), 105-113

Ali, S. M., Gupta, N., Nayak, G. K., & Lenka, R. K. (2016, December). Big data visualization: Tools and challenges. In *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)* (pp. 656-660). IEEE.

Dimara, E., & Perin, C. (2019). What is interaction for data visualization?. *IEEE transactions on visualization and computer graphics*, *26*(1), 119-129.

Du, M., & Yuan, X. (2020). A survey of competitive sports data visualization and visual analysis. *Journal of Visualization*. https://doi.org/10.1007/s12650-020-00687-2

Few, S. (2007). Data Visualization - Past, Present, and Future. https://www.perceptualedge.com/articles/Whitepapers/Data_Visualization.pdf

Gandhi, P., & Pruthi, J. (2020). Data Visualization Techniques: Traditional Data to Big Data. In *Data Visualization* (pp. 53-74). Springer, Singapore.

john. (2022, February 18). *Understanding Scrum*. Institute of Project Management. https://www.projectmanagement.ie/blog/understanding-scrum/

Peek, S. (2013, August 26). *What Is Agile Scrum Methodology?* Business News Daily; businessnewsdaily.com. https://www.businessnewsdaily.com/4987-what-is-agile-scrum-methodology.html

Perin, C., Vuillemot, R., Stolper, C. D., Stasko, J. T., Wood, J., & Carpendale, S. (2018). State of the Art of Sports Data Visualization. *Computer Graphics Forum*, *37*(3), 663–686. https://doi.org/10.1111/cgf.13447

Qin, X., Luo, Y., Tang, N., & Li, G. (2019). Making data visualization more efficient and effective: a survey. *The VLDB Journal*, *29*(1), 93–117. https://doi.org/10.1007/s00778-019-00588-3

17 Important Data Visualization Techniques | HBS Online. (2019, September 17). Business Insights Blog. https://online.hbs.edu/blog/post/data-visualization-techniques

10 Appendices

10.1 Requirements Survey

Do you consider yourself a football fan

- Yes
- No

How often do you watch football

- Very Often
- Often
- Not often
- Never

What leagues do you watch most often

- Premier League
- Bundesliga
- La Liga
- Serie A
- Ligue 1
- All
- Other

Are you interested in the use of Data in football

- ○ Yes
- ○ No

Do you regularly seek data visualisations on football data

- ○ Yes
- ○ No

What data do you look out for

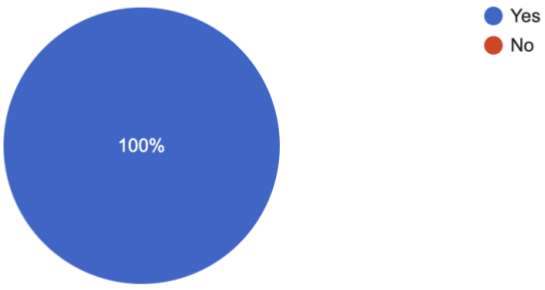
- ☐ Player -Level Data
- ☐ Team-Level Data
- ☐ League- Level Data

10.2 User Testing Survey

Were you easily able to navigate to the different pages of the application.

Copy

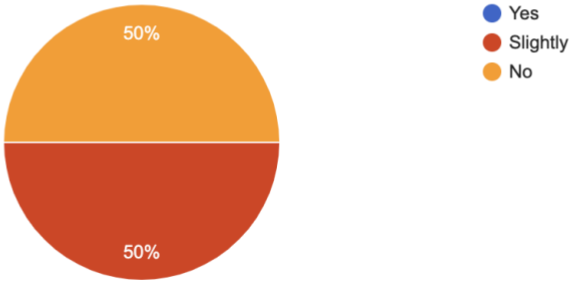
4 responses



On the Player Scatter Plot pages, did you have trouble understanding how to use the different inputs and filters

Copy

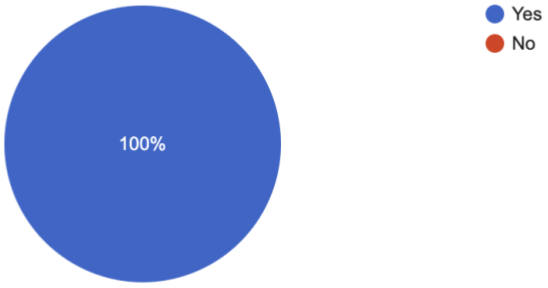
4 responses



Were you easily able to find a certain player or players in the plot

Copy

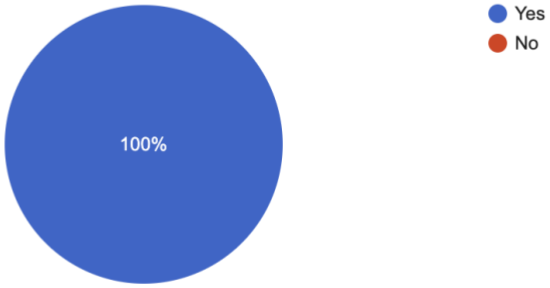
4 responses



Was the Pizza Chart Page easy to understand

Copy

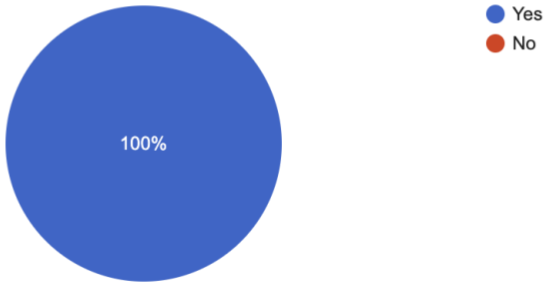
4 responses



On the Pizza Chart page, was it useful for understanding how well a player is performing overall

Copy

4 responses



On the Pizza Chart page, was it easy to compare two players and was the chart easy to read

Copy

4 responses

What features did you like (that would encourage you to use the application again)

4 responses

Were there any features you thought the application should have had but didn't

4 responses

How likely are you to use the application again

Copy

4 responses