STATSBOMB

# Accessing & Working With StatsBomb Data In R

## What is R and Why Use It?

## What Is R and Why Use It?

R is a programming language that is useful for managing large datasets.
It is especially useful in the world of football data, as it allows us to manipulate that data to various ends. Such as creating metrics out of the data and visualising it.

R itself can be downloaded here:

https://cran.r-project.org/mirrors.html

We at StatsBomb use R regularly (amongst other coding languages) in day-to-day work, particularly within our analysis department. Spreadsheets are a viable route when you're just starting out, but eventually the datasets become too big and unwieldy, performing nuanced dissection of them becomes too complicated.

Once you've gotten over the learning curve, R is ideal for parsing data and working with it however you like in a fast manner.

## RStudio

The base version of R is a somewhat cumbersome piece of software. This has lead to the creation of many different 'IDE's (integrated development environment). These are wrappers around the initial R install that make most tasks within R easier and more manageable for the end user. The most popular of these is RStudio:

https://www.rstudio.com/products/rstudio/

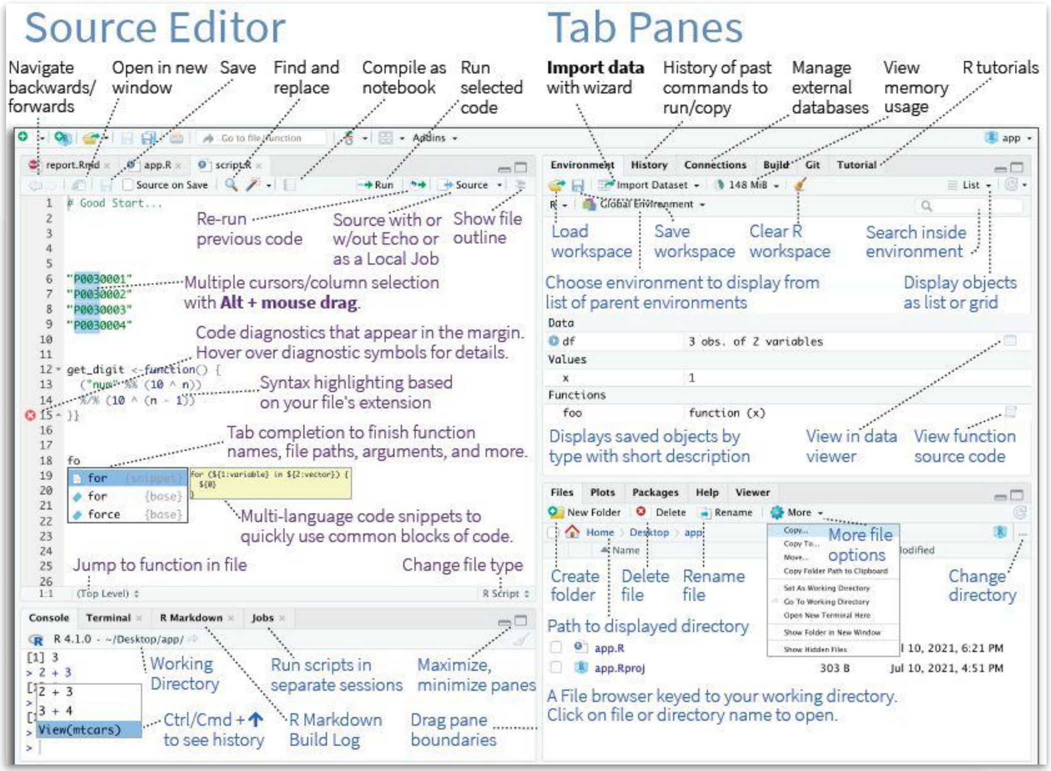
It is recommended that you install RStudio (or any similar IDE that you find and prefer) as most users do. It will make working with StatsBomb's data a cleaner, simpler process.

# Opening a New R *Project*

This (minus the annotations of course) is what you should see when you load up R Studio.

If you're wondering what a particular option or section of R Studio does, then there's a handy set of cheat sheets for it and many other R-related subjects at **[https://www.rstudio.com/resources/cheatsheets/](https://www.rstudio.com/resources/cheatsheets/)**

There are a well of resources out there with detailed answers to just about any question you could have related to R.



Source Editor
Tab Panes
Navigate backwards/ forwards
Open in new window
Save
Find and replace
Compile as notebook
Run selected code
Import data with wizard
History of past commands to run/copy
Manage external databases
View memory usage
R tutorials
Re-run previous code
Source with or w/out Echo or as a Local Job
Show file outline
Load workspace
Save workspace
Clear R workspace
Search inside environment
Multiple cursors/column selection with Alt + mouse drag.
Choose environment to display from list of parent environments
Display objects as list or grid
Code diagnostics that appear in the margin. Hover over diagnostic symbols for details.
Syntax highlighting based on your file's extension
Tab completion to finish function names, file paths, arguments, and more.
Displays saved objects by type with short description
View in data viewer
View function source code
Multi-language code snippets to quickly use common blocks of code.
Jump to function in file
Change file type
More file options
Create folder
Delete file
Rename file
Change directory
Path to displayed directory
Working Directory
Run scripts in separate sessions
Maximize, minimize panes
Ctrl/Cmd + ↑ to see history
R Markdown Build Log
Drag pane boundaries
A File browser keyed to your working directory. Click on file or directory name to open.
report.Rmd
app.R
script.R
Source on Save
Run
Source
Go to file/function
Addins
Environment
History
Connections
Build
Git
Tutorial
Import Dataset
148 MiB
List
Global Environment
Data
Values
Functions
Files
Plots
Packages
Help
Viewer
New Folder
Delete
Rename
More
Home
Desktop
app
Name
Modified
Copy...
Copy To...
Move...
Copy Folder Path to Clipboard
Set As Working Directory
Go To Working Directory
Open New Terminal Here
Show Folder in New Window
Show Hidden Files
Console
Terminal
R Markdown
Jobs
(Top Level)
R Script

```
# Good Start...

"P0030001"
"P0030002"
"P0030003"
"P0030004"

get_digit <- function() {
  ("num" %% (10 ^ n))
  %/% (10 ^ (n - 1))
}}

fo
for     {snippet}   for (${1:variable} in ${2:vector}) {
for     {base}        ${0}
force   {base}      }
```

```
df      3 obs. of 2 variables
x       1
foo     function (x)
```

```
app.R                   Jul 10, 2021, 6:21 PM
app.Rproj      303 B    Jul 10, 2021, 4:51 PM
```

```
R 4.1.0 · ~/Desktop/app/
[1] 3
> 2 + 3
[1] 2 + 3
> 3 + 4
> View(mtcars)
>
```

STATSBOMB

## Packages & ‘StatsBombR’

# What is an ‘R Package’?

‘Packages’ are downloadable bundles of functions that make tasks easier. Most packages are installed by running *install.packages(‘PackageNameHere’)*. However, if the package comes via Github we use the *devtools* package to install it (this includes StatsBombR, which we will walk through installing on the next page).

**The main packages we will focus on here and which need installing are:**
**‘tidyverse’:** tidyverse contains a whole host of other packages (such as dplyr and magrittr) that are useful for manipulating data. *install.packages(“tidyverse”)*

**‘devtools’:** Most packages are hosted on CRAN. However there are also countless useful ones hosted on Github. Devtools allows for downloading of packages directly from Github. *install.packages(“devtools”)*

**‘ggplot2’:** The most popular package for visualising data within R. It is contained within tidyverse.

‘**StatsBombR’:** This is StatsBomb’s own package for parsing our data.

Once a package is installed it can be loaded into R by running *library(PackageNameHere)*. You should load all of these at the start of any session.

## What is 'StatsBombR' and how to Install It?

StatsBomb's former data scientist Derrick Yam created 'StatsBombR', an R package dedicated to making using StatBomb's data in R much easier. It can be found on Github at the following link, along with much more information on its uses. There are lots of helpful functions within it that you should get to know.

https://github.com/statsbomb/StatsBombR

To install the package in R, you'll need to install the 'Devtools' package, which can be done by running the following line of code:

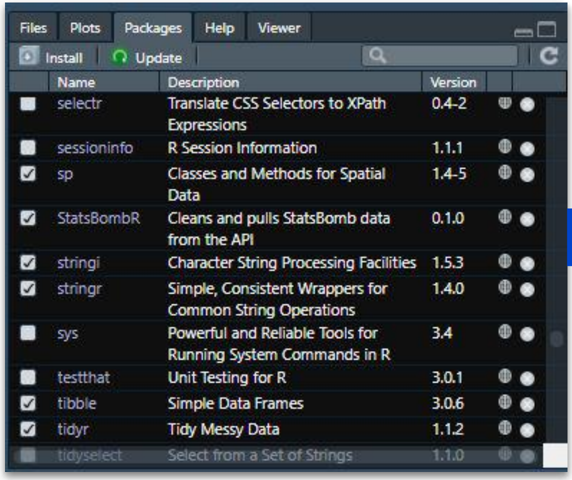
```
install.packages("devtools")
install.packages("remotes")
remotes::install_version("SDMTools", "1.1-221")
```

Then, to install StatsBombR itself, run:

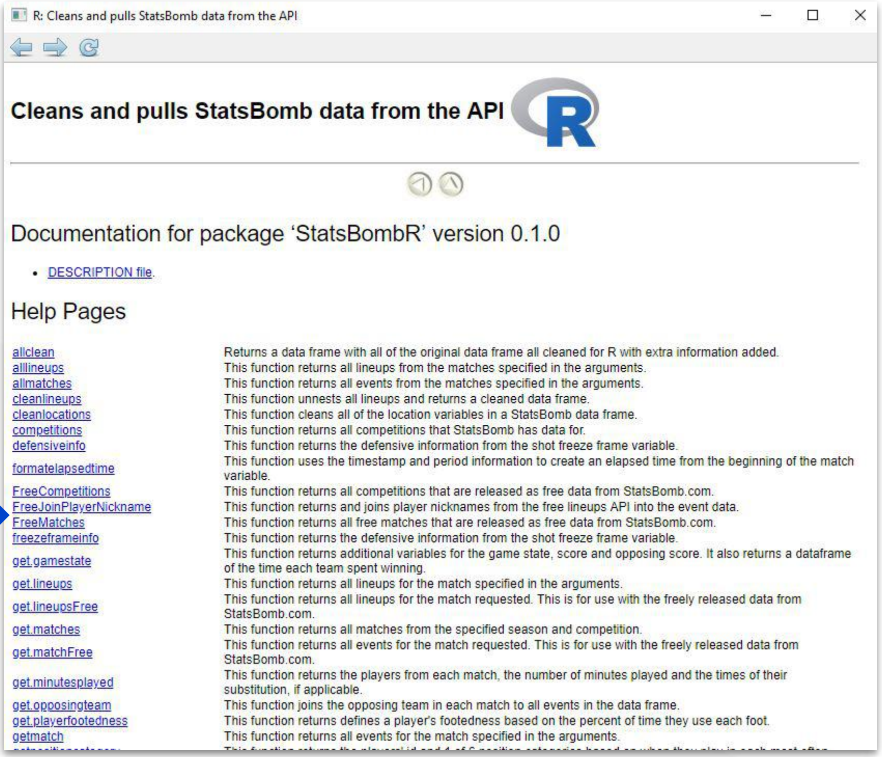
```
devtools::install_github("statsbomb/StatsBombR")
```

# Finding More Info On Packages

If you want more detail on the various functions within a package then click on the package's name in the viewer in the bottom right. That will take you to the documentation for that package. It should contain all sorts of information on the ins and outs of its functions.



| | Name | Description | Version |
|---|---|---|---|
| ☐ | selectr | Translate CSS Selectors to XPath Expressions | 0.4-2 |
| ☐ | sessioninfo | R Session Information | 1.1.1 |
| ✓ | sp | Classes and Methods for Spatial Data | 1.4-5 |
| ✓ | StatsBombR | Cleans and pulls StatsBomb data from the API | 0.1.0 |
| ✓ | stringi | Character String Processing Facilities | 1.5.3 |
| ✓ | stringr | Simple, Consistent Wrappers for Common String Operations | 1.4.0 |
| ☐ | sys | Powerful and Reliable Tools for Running System Commands in R | 3.4 |
| ☐ | testthat | Unit Testing for R | 3.0.1 |
| ✓ | tibble | Simple Data Frames | 3.0.6 |
| ✓ | tidyr | Tidy Messy Data | 1.1.2 |
| ☐ | tidyselect | Select from a Set of Strings | 1.1.0 |



Cleans and pulls StatsBomb data from the API

Documentation for package 'StatsBombR' version 0.1.0

- DESCRIPTION file.

Help Pages

| | |
|---|---|
| allclean | Returns a data frame with all of the original data frame all cleaned for R with extra information added. |
| alllineups | This function returns all lineups from the matches specified in the arguments. |
| allmatches | This function returns all events from the matches specified in the arguments. |
| cleanlineups | This function unnests all lineups and returns a cleaned data frame. |
| cleanlocations | This function cleans all of the location variables in a StatsBomb data frame. |
| competitions | This function returns all competitions that StatsBomb has data for. |
| defensiveinfo | This function returns the defensive information from the shot freeze frame variable. |
| formatelapsedtime | This function uses the timestamp and period information to create an elapsed time from the beginning of the match variable. |
| FreeCompetitions | This function returns all competitions that are released as free data from StatsBomb.com. |
| FreeJoinPlayerNickname | This function returns and joins player nicknames from the free lineups API into the event data. |
| FreeMatches | This function returns all free matches that are released as free data from StatsBomb.com. |
| freezeframeinfo | This function returns the defensive information from the shot freeze frame variable. |
| get.gamestate | This function returns additional variables for the game state, score and opposing score. It also returns a dataframe of the time each team spent winning. |
| get.lineups | This function returns all lineups for the match specified in the arguments. |
| get.lineupsFree | This function returns all lineups for the match requested. This is for use with the freely released data from StatsBomb.com. |
| get.matches | This function returns all matches from the specified season and competition. |
| get.matchFree | This function returns all events for the match requested. This is for use with the freely released data from StatsBomb.com. |
| get.minutesplayed | This function returns the players from each match, the number of minutes played and the times of their substitution, if applicable. |
| get.opposingteam | This function joins the opposing team in each match to all events in the data frame. |
| get.playerfootedness | This function returns defines a player's footedness based on the percent of time they use each foot. |
| getmatch | This function returns all events for the match specified in the arguments. |

## Pulling in StatsBomb Data

STATSBOMB

# Key Functions for Getting the Free Data

There are several key functions within StatsBombR to familiarise yourself with for bringing StatsBomb Data into R.

***FreeCompetitions()*** - This shows you all the competitions that are available as free data

If you want to store the output of this (or any other functions) so you can pull it up at any time, instead of just having it in the R console, you can run something like the following:

*Comp <- FreeCompetitions().* Then, anytime you run Comp (or whatever word you choose to store it under, you can go with anything), you will see the output of *FreeCompetitions().*

***Matches <- FreeMatches(Comp)*** - This shows the available matches within the competitions chosen

***StatsBombData <- free_allevents(MatchesDF = Matches, Parallel = T)*** - This pulls all the event data for the matches that are chosen.

## Pulling the Free Data

Now we're going to run through an example of how to pull the data into R. Open up a new 'script', so we can store this code and have it easily accessible, by going to File -> New File -> R Script. This script can be saved at any time.

```
library(tidyverse)
library(StatsBombR) #1

Comp <- FreeCompetitions() %>%
filter(competition_id==37 & season_name=="2020/2021") #2

Matches <- FreeMatches(Comp) #3

StatsBombData <- free_allevents(MatchesDF = Matches, Parallel = T) #4

StatsBombData = allclean(StatsBombData) #5
```

**#1**: *tidyverse* loads many different packages. Most important for this task are dplyr and magrittr. *StatsBombR* loads StatsBombR.

**#2**: This grabs the competitions that are available to the user and filters it down, using dplyr's 'filter' function, to just the 2020/21 FA Women's Super League season in this example.

**#3**: This pulls all the matches for the desired competition.

**#4**: Now we have created a 'dataframe' (essentially a table) called 'StatsBombData' (or whatever you choose to call it) of the free event data for the FAWSL season in 2020/2021.

**#5**: Extracts lots of relevant information such as x/y coordinates. More information can be found in the package info. Be sure to familiarise yourself with the columns it creates using *names(nameofyourdfhere)*.

## Working With the Data

STATSBOMB

## Getting to Know the Data

On our Github page - where our free data is hosted - we have put the specification documents for StatsBomb Data. These are available to view or download at any time and will hopefully answer any questions you may have about what a certain event type is or any similar inquiries.

These documents include:

**Open Data Competitions v2.0.0.pdf** - Covers the objects contained within the competitions information *( FreeCompetitions() )*.

**Open Data Matches v3.0.0.pdf** - Describes the match info download *( FreeMatches() )*.

**Open Data Lineups v2.0.0.pdf** - Describes the structure of the lineup info *( getlineupsFree() )*.

**Open Data Events v4.0.0.pdf** - Explains the meaning of the column names within the event data.

**StatsBomb Event Data Specification v1.1.pdf** - The full breakdown of all the events within the data.

# Data Use Cases

Now that we have our StatsBombData file, we're going to run through some ways you can use the data and familiarise yourself with R in the process. There will be four use cases, increasing in complexity as they go:

**Use Case 1: Shots and Goals** - A simple but important starting point. Here we will extract shots and goals totals for each team, then look at how to do the same but on a per game basis.

**Use Case 2: Graphing Shots On a Chart** - After we have the shots and goals data, how can we take that and create a starter chart from it?

**Use Case 3: Player Shots Per 90** - Getting shots for players is simple enough after doing so for teams. But then how can we adjust those figures on a per 90 basis?

**Use Case 4: Mapping Passes** - Filtering our data down to just a subset of passes and then using R's ggplot2 to plot those passes on a pitch.

## Data Use Case 1: Goals and Shots

```
shots_goals = StatsBombData %>%
group_by(team.name) %>% #1
summarise(shots = sum(type.name=="Shot", na.rm = TRUE),
goals = sum(shot.outcome.name=="Goal", na.rm = TRUE)) #2
```

Let's break that down bit by bit:

***#1***: This code groups the data by team, so that whatever operation we perform on it will be done on a team by team basis. I.e, we will find the shots and goals for every team one by one.

***#2***: *Summarise* takes whatever operation we give it and produces a new, separate table out of it. The vast majority of *summarise* uses come after *group_by*.

*shots = sum(type.name=="Shot", na.rm = TRUE)* is telling it to create a new column called 'shots' that sums up all the rows under the 'type.name' column that contain the word *"Shot". na.rm = TRUE* tells it to ignore any NAs within that column.

*shot.outcome.name=="Goal", na.rm = TRUE)* does the same but for goals.

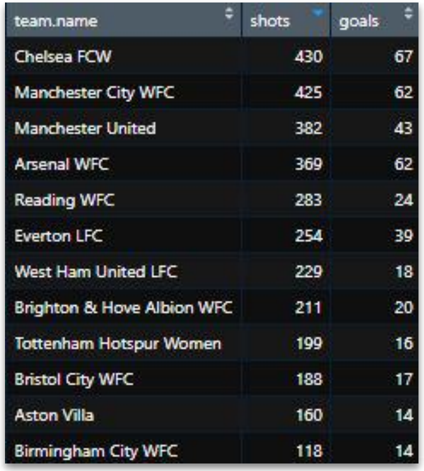
## Data Use Case 1: Goals and Shots

You should now have a table that looks like this.

If we want to do this same calculation but on a per game basis, we can change it to:

```
shots_goals = StatsBombData %>%
group_by(team.name) %>%
summarise(shots = sum(type.name=="Shot", na.rm = TRUE)/n_distinct(match_id),
goals = sum(shot.outcome.name=="Goal", na.rm = TRUE)/n_distinct(match_id))
```

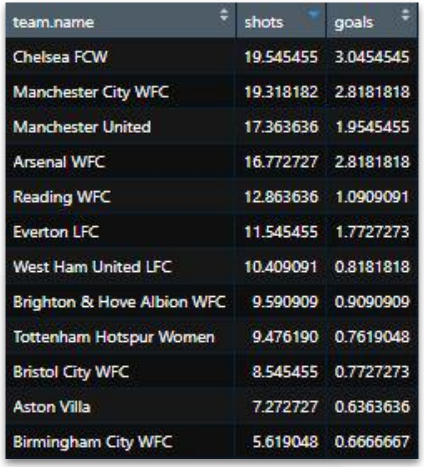
Adding in the '*n_distinct(match_id)*' means we are dividing the number of shots/goals by each distinct (or unique) instance of a match, for every team. I.e, we are dividing the numbers per game.



| team.name | shots | goals |
|---|---|---|
| Chelsea FCW | 430 | 67 |
| Manchester City WFC | 425 | 62 |
| Manchester United | 382 | 43 |
| Arsenal WFC | 369 | 62 |
| Reading WFC | 283 | 24 |
| Everton LFC | 254 | 39 |
| West Ham United LFC | 229 | 18 |
| Brighton & Hove Albion WFC | 211 | 20 |
| Tottenham Hotspur Women | 199 | 16 |
| Bristol City WFC | 188 | 17 |
| Aston Villa | 160 | 14 |
| Birmingham City WFC | 118 | 14 |

**Totals**

**Per Game**



| team.name | shots | goals |
|---|---|---|
| Chelsea FCW | 19.545455 | 3.0454545 |
| Manchester City WFC | 19.318182 | 2.8181818 |
| Manchester United | 17.363636 | 1.9545455 |
| Arsenal WFC | 16.772727 | 2.8181818 |
| Reading WFC | 12.863636 | 1.0909091 |
| Everton LFC | 11.545455 | 1.7727273 |
| West Ham United LFC | 10.409091 | 0.8181818 |
| Brighton & Hove Albion WFC | 9.590909 | 0.9090909 |
| Tottenham Hotspur Women | 9.476190 | 0.7619048 |
| Bristol City WFC | 8.545455 | 0.7727273 |
| Aston Villa | 7.272727 | 0.6363636 |
| Birmingham City WFC | 5.619048 | 0.6666667 |

STATSBOMB

## Data Use Case 2: From Data to a Chart

```
library(ggplot2)

ggplot(data = shots_goals,
    aes(x = reorder(team.name, shots), y = shots)) + #1
geom_bar(stat = "identity", width = 0.5) + #2
labs(y="Shots") + #3
theme(axis.title.y = element_blank()) + #4
scale_y_continuous( expand = c(0,0)) + #5
coord_flip() + #6
theme_SB() #7
```

***#1:*** Here we are telling ggplot what data we are using and what we want to plot on the x/y axes of our chart. 'Reorder' quite literally reorders the team names according to the number of shots they have.

***#2:*** Now we are telling ggplot to format it is a bar chart.

***#3:*** This relabels the shots axis.

***#4:*** This removes the title for the axis.

***#5:*** Here we cut down on the space between the bars and the edge of the plot

***#6:*** This flips the entire plot, with the bars now going horizontally instead.

***#7:*** *theme_SB()* is our own internal visual aesthetic for ggplot charts that we have packaged with StatsBombR. Optional of course.

# Data Use Case 2: From Data to a Chart

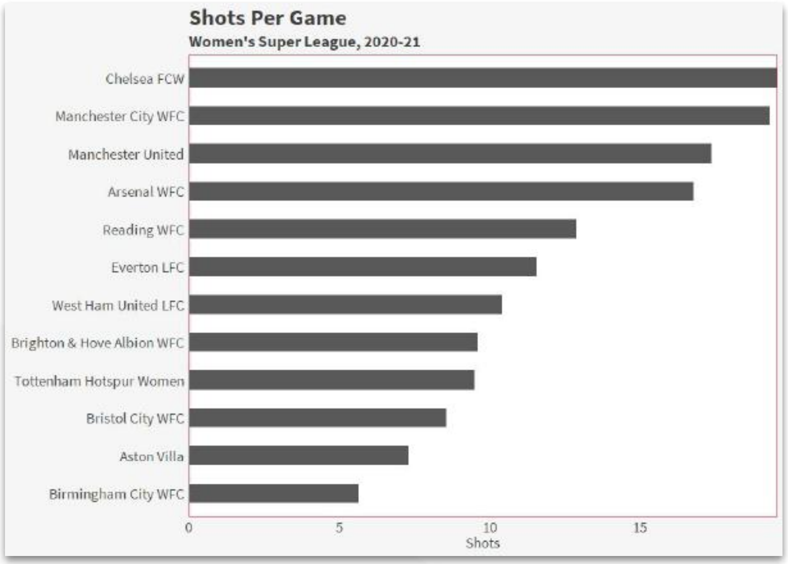
All that should result in a chart like this.

Of course this is a basic chart, fairly visually plain on its own and it could be altered in many ways to add your own spin on it.

Almost every element of a ggplot chart - from the text to the plotted data itself and beyond - can be changed how you see fit. There's lots of room for creativity.

For an in depth reference point on what kind of charts you can create or how you can modify them, you can look here:

[https://ggplot2.tidyverse.org/reference/](https://ggplot2.tidyverse.org/reference/)



STATSBOMB

# Data Use Case 3: Player Shots Per 90

```
player_shots = StatsBombData %>%
group_by(player.name, player.id) %>%
summarise(shots = sum(type.name=="Shot", na.rm = TRUE)) #1

player_minutes = get.minutesplayed(StatsBombData) #2

player_minutes = player_minutes %>%
group_by(player.id) %>%
summarise(minutes = sum(MinutesPlayed)) #3

player_shots = left_join(player_shots, player_minutes) #4

player_shots = player_shots %>% mutate(nineties = minutes/90) #5

player_shots = player_shots %>% mutate(shots_per90 = shots/nineties) #6
```

***#1:*** Much the same as the team calculation. We are including 'player.id' here as it will be important later.

***#2:*** This function gives us the minutes played in each match by ever player in the dataset.

***#3:*** Now we group that by player and sum it altogether to get their total minutes played.

***#4:*** *left_join* allows us to combine our shots table and our minutes table, with the the player.id acting as a reference point.

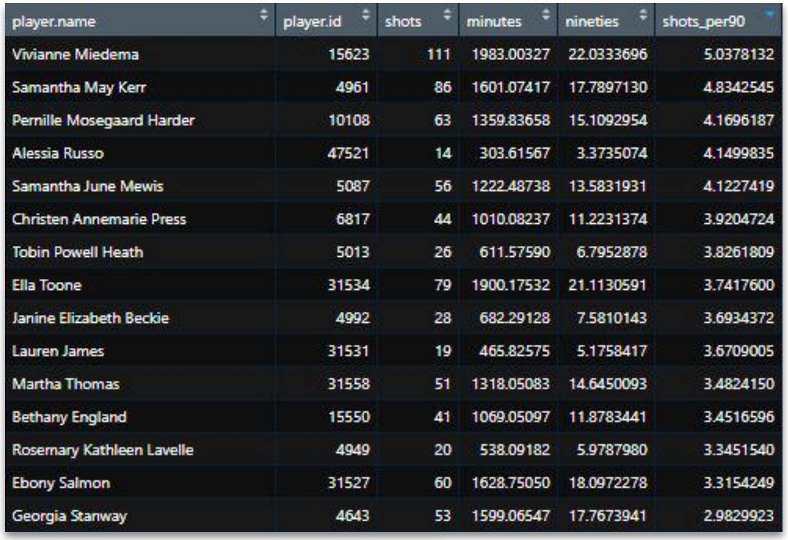
***#5:*** *mutate* is a dplyr function that creates a new column. In this instance we are creating a column that divides the minutes totals by 90, giving us each playerÈs number of 90s played for the season.

***#6:*** Finally we divide our shots totals by our number of 90s to get our shots per 90s column.

## Data Use Case 3: Player Shots Per 90

Now you'll have shots per 90 for all the players across the WSL (or your league of choice). This can be cleaned up using dplyr's 'filter' function, in order to get rid of the players with few minutes played.

This same process can of course be applied to all sorts of events with StatsBomb Data. Certain types of passes, defensive actions and so on.



| player.name | player.id | shots | minutes | nineties | shots_per90 |
|---|---|---|---|---|---|
| Vivianne Miedema | 15623 | 111 | 1983.00327 | 22.0333696 | 5.0378132 |
| Samantha May Kerr | 4961 | 86 | 1601.07417 | 17.7897130 | 4.8342545 |
| Pernille Mosegaard Harder | 10108 | 63 | 1359.83658 | 15.1092954 | 4.1696187 |
| Alessia Russo | 47521 | 14 | 303.61567 | 3.3735074 | 4.1499835 |
| Samantha June Mewis | 5087 | 56 | 1222.48738 | 13.5831931 | 4.1227419 |
| Christen Annemarie Press | 6817 | 44 | 1010.08237 | 11.2231374 | 3.9204724 |
| Tobin Powell Heath | 5013 | 26 | 611.57590 | 6.7952878 | 3.8261809 |
| Ella Toone | 31534 | 79 | 1900.17532 | 21.1130591 | 3.7417600 |
| Janine Elizabeth Beckie | 4992 | 28 | 682.29128 | 7.5810143 | 3.6934372 |
| Lauren James | 31531 | 19 | 465.82575 | 5.1758417 | 3.6709005 |
| Martha Thomas | 31558 | 51 | 1318.05083 | 14.6450093 | 3.4824150 |
| Bethany England | 15550 | 41 | 1069.05097 | 11.8783441 | 3.4516596 |
| Rosemary Kathleen Lavelle | 4949 | 20 | 538.09182 | 5.9787980 | 3.3451540 |
| Ebony Salmon | 31527 | 60 | 1628.75050 | 18.0972278 | 3.3154249 |
| Georgia Stanway | 4643 | 53 | 1599.06547 | 17.7673941 | 2.9829923 |

## Data Use Case 4: Plotting Passes

Finally, we're going to look at plotting a player's passes on a pitch. For this we of course need some sort of pitch visualisation. You might want to create your own once you become more familiar with ggplot and using it for more complex purposes (there will be a flexible version that we use later in this presentation). However, handily, there are several pre-made solutions out there.

The one we'll be using here comes courtesy of **FC rStats.** A twitter user who has put together various helpful, public R packages for parsing football data. The package is called **'SBPitch'** and it does exactly what it says on the tin. There will be further options in the 'Other Useful Packages' at the end of this document. First let's get it installed with the following code:

*devtools::install_github("FCrSTATS/SBpitch")*

We're going to plot Fran Kirby's completed passes into the box for the 2020/21 FA Women's Super League season. Plotting all of her passes would get messy of course, so this is a clearer subset. Make sure you've used the functions previously discussed to pull that data.

## Data Use Case 4: Plotting Passes

```
library(SBpitch)

passes = wsldata %>%
  filter(type.name=="Pass" & is.na(pass.outcome.name) &
player.id==4641) %>% #1
  filter(pass.end_location.x>=102 & pass.end_location.y<=62 &
pass.end_location.y>=18) #2

create_Pitch() +
  geom_segment(data = passes, aes(x = location.x, y = location.y,
                              xend = pass.end_location.x, yend = pass.end_location.y),
            lineend = "round", size = 0.5, colour = "#000000", arrow =
arrow(length = unit(0.07, "inches"), ends = "last", type = "open")) + #3
  labs(title = "Fran Kirby, Completed Box Passes", subtitle = "WSL,
2020-21") + #4
scale_y_reverse() + #5
coord_fixed(ratio = 105/100) #6
```

***#1:*** Pull some of the FA WSL data of your choice and call it 'wsldata' for us to work with here. Then we can filter to Fran Kirby's passes. *is.na(pass.outcome.name)* filters to only completed passes.

***#2:*** Filtering to passes within the box. The coordinates for pitch markings in SBD can be found in our **event spec.**

***#3:*** This creates an arrow from one point (location.x/y, the start part of the pass) to an end point (pass.end_location.x/y, the end of the pass). *Lineend, size* and *length* are are all customisation options for the arrow.

***#4:*** Creates a title and a subtitle for the plot. You can also add captions using *caption* =, along with other options.

***#5:*** Reverses the y axis. Otherwise the data would be plotted on the wrong side of the pitch.

***#6:*** Fixes the plot to a certain aspect ratio of your choice, so it doesn't look stretched.

STATSBOMB

# Data Use Case 4: Plotting Passes

You'll have this plot. Again, it's simple and bare but it starts you off and from here you can layer on all sorts of customisation.

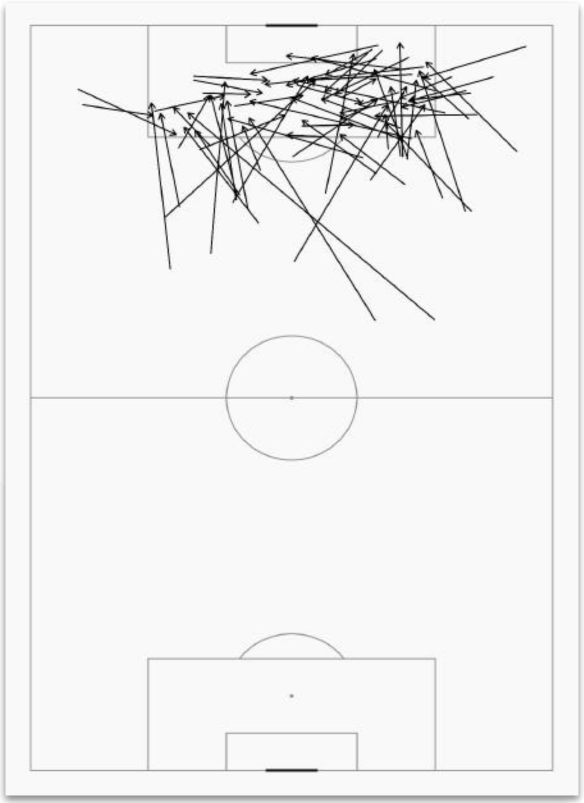
*theme()* options allow you to change the size, placement, font and much more of the titles. As well as to alter lots of other aesthetic aspects of the plot.

You can add *colour =* to *geom_segment()* in order to colour passes according to what you choose.

Again, be sure to dig into the ggplot documentation to get the full scope of how powerful it is.

**This is a great cheat sheet for various ways you can use the package.**

Fran Kirby
Completed Box Passes, WSL 2020-21



# Useful StatsBombR Functions

There all sorts of functions within StatsBombR for different purposes. You can find them all **here**, deeper into the github page. Not all functions are related to the free data. Some are only accessible for customers via our API. Here's a quick rundown of some you may find useful:

***allclean()*** - Mentioned previously but to elucidate: this extrapolates lots of new, helpful columns from the pre existing columns. For example, it takes the location column and splits it up into separate x/y columns. It also extracts freeze frame data and goalkeeper information. Make sure to use.

***get.playerfootedness()*** - Gives you a player's assumed preferred foot using our pass footedness data.

***get.opposingteam()*** - Returns an opposing team column for each team in each match.

***get.gamestate()*** - Returns information for how much time each team spent in various game states (winning/drawing/losing) for each match.

***annotate_pitchSB()*** - Our own solution for plotting a pitch with ggplot.

## Other Useful Packages

The community around R is packed with packages that fulfill all sorts of needs. Chances are that, if you’re looking to do something in R or fix some sort of issue, there’s a package out there for it. There are far too many to name but here’s a brief selection of some that may be relevant to working with StatsBomb Data:

**Ben Torvaney, ggsoccer** - A package that contains an alternative for plotting a pitch with SB Data.

**Joe Gallagher, soccermatics** - Also offers an option for pitch plotting along with other useful shortcuts for creating heatmaps and so on.

**ggrepel** - Useful for when you’re having issues with overlapping labels on a chart.

**gganimate** - If you ever feel like getting more elaborate with your graphics, this gives you a simple way to create animated ones within R and ggplot.

## Doing More With StatsBomb Data In R

## More Data Use Cases

The content beyond this point in the guide is aimed at those that have been through the first part of the guide and have been playing about with SBD for a while now.

It's important that you have done this first as we will not be walking through absolutely *everything* -- we assume a certain level of familiarity with R in this section.

There'll be three use cases this time:

**Use Case 5: xG Assisted, Joining, and xG+xGA** - An example of how to create and then plot custom metrics with the data, creating xG Assisted in a dataframe using 'joining' and then creating an xG + xG Assisted plot.

**Use Case 6: Graphing Shots On a Chart** - Heatmaps are one of the everpresents in football data. They are fairly easy to make in R once you get your head round how to do so, but can be unintuitive without having it explained first.

**Use Case 7: Shot Maps** - Another of the quintessential football visualisations, shot maps come in many shapes and sizes with an inconsistent overlap in design language between them. This version will attempt to give you the basics.

## Data Use Case 5: xG Assisted, Joining, and xG+xGA

STATSBOMB

# Data Use Case 5: xG Assisted, Joining, and xG+xGA

xG assisted does not exist in our data initially. However, given that xGA is the xG value of a shot that a key pass/assist created, and that xG values do exist in our data, we can create xGA quite easily via *joining*.

```
library(tidyverse)
library(StatsBombR)
xGA = events %>%
filter(type.name=="Shot") %>% #1
select(shot.key_pass_id, xGA = shot.statsbomb_xg) #2

shot_assists = left_join(events, xGA, by = c("id" = "shot.key_pass_id"))
%>% #3
select(team.name, player.name, player.id, type.name, pass.shot_assist,
pass.goal_assist, xGA ) %>% #4
filter(pass.shot_assist==TRUE | pass.goal_assist==TRUE) #5
```

**#1** Filtering the data to just shots, as they are the only events with xG values.

**#2** *Select()* allows you to choose which columns you want to, well, select, from your daata, as not all are always necessary - especially with big datasets. First we are selecting the *shot.key_pass_id* column, which is a variable attached to shots that is just the ID of the pass that created the shot. You can also rename columns within *select()* which is what we are doing with *xGA = shot.statsbomb_xg*. This is so that, when we join it with the passes, it already has the correct name.

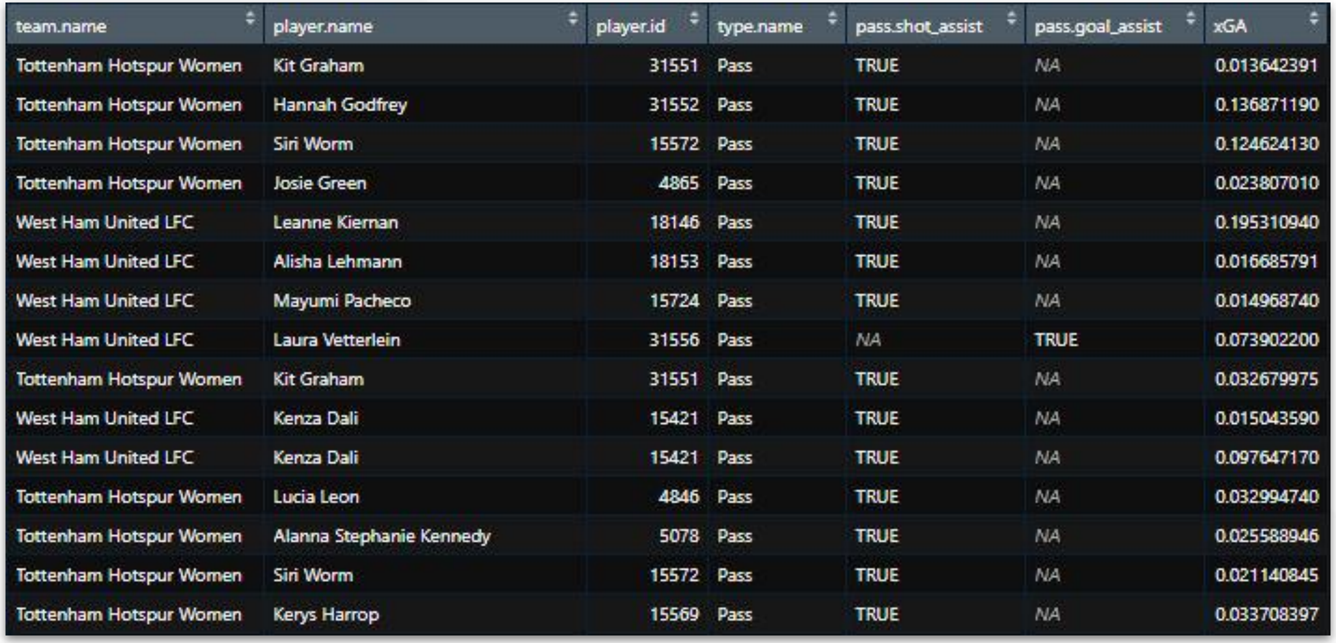
**#3** *left_join()* lets you combine the columns from two different DFs by using two columns within either side of the join as reference keys. So in this example we are taking our initial DF ('events') and joining it with the one we just made ('xGA'). The key is the *by = c("id" = "shot.key_pass_id")* part, this is saying 'join these two DFs on instances where the id column in events matches the 'shot.key_pass_id' column in xGA'. So now the passes have the xG of the shots they created attached to them under the new column 'xGA'.

**#4** Again selecting just the relevant columns.

**#5** Filtering our data down to just key passes/assists.

## Data Use Case 5: xG Assisted, Joining, and xG+xGA

The end result should look like this:



| team.name | player.name | player.id | type.name | pass.shot_assist | pass.goal_assist | xGA |
|---|---|---|---|---|---|---|
| Tottenham Hotspur Women | Kit Graham | 31551 | Pass | TRUE | NA | 0.013642391 |
| Tottenham Hotspur Women | Hannah Godfrey | 31552 | Pass | TRUE | NA | 0.136871190 |
| Tottenham Hotspur Women | Siri Worm | 15572 | Pass | TRUE | NA | 0.124624130 |
| Tottenham Hotspur Women | Josie Green | 4865 | Pass | TRUE | NA | 0.023807010 |
| West Ham United LFC | Leanne Kiernan | 18146 | Pass | TRUE | NA | 0.195310940 |
| West Ham United LFC | Alisha Lehmann | 18153 | Pass | TRUE | NA | 0.016685791 |
| West Ham United LFC | Mayumi Pacheco | 15724 | Pass | TRUE | NA | 0.014968740 |
| West Ham United LFC | Laura Vetterlein | 31556 | Pass | NA | TRUE | 0.073902200 |
| Tottenham Hotspur Women | Kit Graham | 31551 | Pass | TRUE | NA | 0.032679975 |
| West Ham United LFC | Kenza Dali | 15421 | Pass | TRUE | NA | 0.015043590 |
| West Ham United LFC | Kenza Dali | 15421 | Pass | TRUE | NA | 0.097647170 |
| Tottenham Hotspur Women | Lucia Leon | 4846 | Pass | TRUE | NA | 0.032994740 |
| Tottenham Hotspur Women | Alanna Stephanie Kennedy | 5078 | Pass | TRUE | NA | 0.025588946 |
| Tottenham Hotspur Women | Siri Worm | 15572 | Pass | TRUE | NA | 0.021140845 |
| Tottenham Hotspur Women | Kerys Harrop | 15569 | Pass | TRUE | NA | 0.033708397 |

## Data Use Case 5: xG Assisted, Joining, and xG+xGA

All lovely. But what if you want to make a chart out of it?
Say you want to combine it with xG to make a handy xG+xGA per90 chart:

```
player_xGA = shot_assists %>%
group_by(player.name, player.id, team.name) %>%
summarise(xGA = sum(xGA, na.rm = TRUE)) #1

player_xG = events %>%
filter(type.name=="Shot") %>%
filter(shot.type.name!="Penalty" | is.na(shot.type.name)) %>%
group_by(player.name, player.id, team.name) %>%
summarise(xG = sum(shot.statsbomb_xg, na.rm = TRUE)) %>%
left_join(player_xGA) %>%
mutate(xG_xGA = sum(xG+xGA, na.rm =TRUE) ) #2

player_minutes = get.minutesplayed(events)

player_minutes = player_minutes %>%
group_by(player.id) %>%
summarise(minutes = sum(MinutesPlayed)) #3
```

```
player_xG_xGA = left_join(player_xG, player_minutes) %>%
mutate(nineties = minutes/90,
xG_90 = round(xG/nineties, 2),
xGA_90 = round(xGA/nineties,2),
xG_xGA90 = round(xG_xGA/nineties,2) ) #4

chart = player_xG_xGA %>%
ungroup() %>%
filter(minutes>=600) %>%
top_n(n = 15, w = xG_xGA90) #5

chart<-chart %>%
select(1, 9:10)%>%
pivot_longer(-player.name, names_to = "variable", values_to = "value") %>%
filter(variable=="xG_90" | variable=="xGA_90") #6
```

STATSBOMB

# Data Use Case 5: xG Assisted, Joining, and xG+xGA

***#1*** Grouping by player and summing their total xGA for the season.

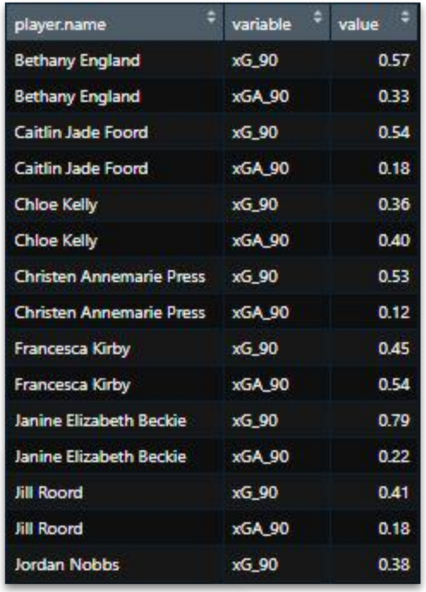
***#2*** Filtering out penalties and summing each player's xG, then joining with the xGA and adding the two together to get a third combined column.

***#3*** Getting minutes played for each player. If you went through the earlier data use cases in this guide you will have done this already.

***#4*** Joining the xG/xGA to the minutes, creating the 90s and dividing each stat by the 90s to get xG per 90 etc.

***#5*** Here we ungroup as we need the data in ungrouped form for what we're about to do. First we filter to players with a minimum of 600 minutes, just to get rid of notably small samples. Then we use *top_n()*. This filters your DF to the top *insert number of your choice here* based on a column you specify. So here we're filtering to the top 15 players in terms of xG90+xGA90.

***#6*** The *pivot_longer()* function flattens out the data. It's easier to explain what that means if you see it first:



| player.name | variable | value |
|---|---|---|
| Bethany England | xG_90 | 0.57 |
| Bethany England | xGA_90 | 0.33 |
| Caitlin Jade Foord | xG_90 | 0.54 |
| Caitlin Jade Foord | xGA_90 | 0.18 |
| Chloe Kelly | xG_90 | 0.36 |
| Chloe Kelly | xGA_90 | 0.40 |
| Christen Annemarie Press | xG_90 | 0.53 |
| Christen Annemarie Press | xGA_90 | 0.12 |
| Francesca Kirby | xG_90 | 0.45 |
| Francesca Kirby | xGA_90 | 0.54 |
| Janine Elizabeth Beckie | xG_90 | 0.79 |
| Janine Elizabeth Beckie | xGA_90 | 0.22 |
| Jill Roord | xG_90 | 0.41 |
| Jill Roord | xGA_90 | 0.18 |
| Jordan Nobbs | xG_90 | 0.38 |

*It's used the player.name as a reference point and creates separate rows for every variable that's left over. We then filter down to just the xG90 and xGA90 variables so now each player has a separate variable and value row for those two metrics.*

## Data Use Case 5: xG Assisted, Joining, and xG+xGA

Now let’s plot it:

```
ggplot(chart, aes(x =reorder(player.name, value), y = value, fill=fct_rev(variable))) + #1
 geom_bar(stat="identity", colour="white")+
 labs(title = "Expected Goal Contribution", subtitle = "FA Women's Super League, 2020-21",
   x="", y="Per 90", caption ="Minimum 600 minutes\nNPxG = Value of shots taken (no penalties)\nxG assisted = Value of shots assisted")+
 theme(axis.text.y = element_text(size=14, color="#333333", family="Source Sans Pro"),
    axis.title = element_text(size=14, color="#333333", family="Source Sans Pro"),
    axis.text.x = element_text(size=14, color="#333333", family="Source Sans Pro"),
    axis.ticks = element_blank(),
    panel.background = element_rect(fill = "white", colour = "white"),
    plot.background = element_rect(fill = "white", colour ="white"),
    panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
    plot.title=element_text(size=24, color="#333333", family="Source Sans Pro" , face="bold"),
    plot.subtitle=element_text(size=18, color="#333333", family="Source Sans Pro", face="bold"),
    plot.caption=element_text(color="#333333", family="Source Sans Pro", size =10),
    text=element_text(family="Source Sans Pro"),
    legend.title=element_blank(),
    legend.text = element_text(size=14, color="#333333", family="Source Sans Pro"),
    legend.position = "bottom") + #2
 scale_fill_manual(values=c("#3371AC", "#DC2228"), labels = c( "xG Assisted","NPxG")) + #3
 scale_y_continuous(expand = c(0, 0), limits= c(0,max(chart$value) + 0.3)) + #4
 coord_flip()+ #5
 guides(fill = guide_legend(reverse = TRUE)) #6
```

STATSBOMB

# Data Use Case 5: xG Assisted, Joining, and xG+xGA

***#1:*** Two things are going on here that are different from your average bar chart. First is *reorder(),* which allows you reorder a variable along either axis based on a second variable. In this instance we are putting the player names on the x axis and reordering them by value - i.e the xG and xGA combined - meaning they are now in descending order from most to least combined xG+xGA. Second is that we've put the 'variable' on the bar fill. This allows us to put two separate metrics onto one bar chart and have them stack, as you will see below, by having them be separate fill colours.

***#2:*** Everything within *labs()* and *theme()* is fairly self explanatory and is just what we have used internally. You can get rid of all this if you like and change it to suit your own design tastes.

***#3:*** Here we are providing specific colour hex codes to the values (so xG = red and xGA = blue) and then labelling them so they are named correctly on the chart's legend.

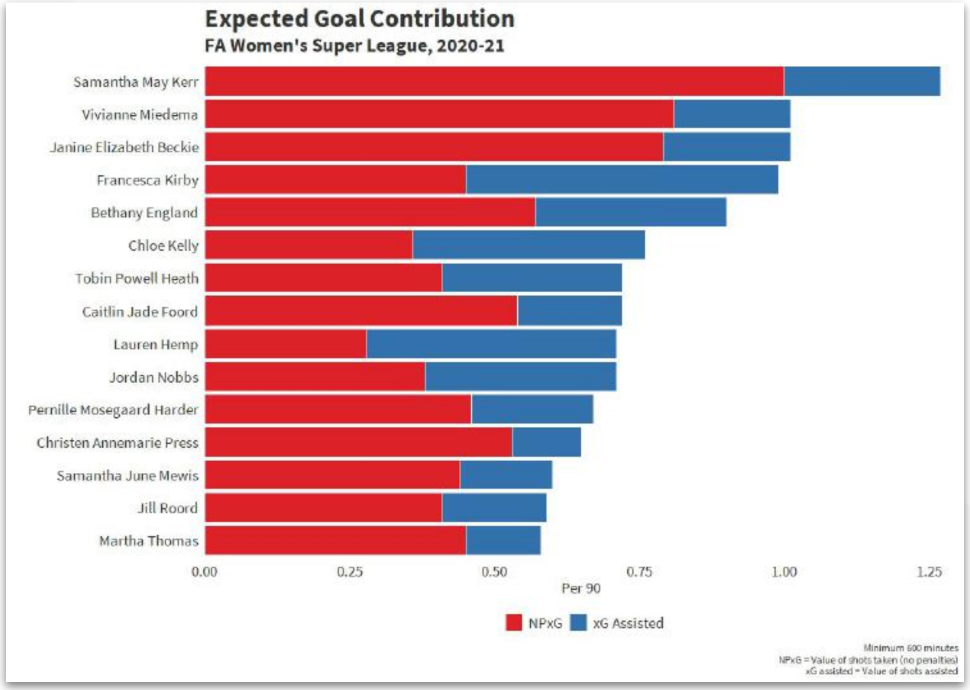
***#4:*** *Expand()* allows you to expand the boundaries of the x or y axis, but if you set the values to (0,0) it also removes all space between the axis and the inner chart itself (if you're having a hard time envisioning that, try removing expand() and see what it looks like). Then we are setting the limits of the y axis so the longest bar on the chart isn't too close to the edge of the chart. '*max(chart$value) + 0.3*' is saying 'take the max value and add 0.3 to make that the upper limit of the y axis'.

***#5:*** Flipping the x axis and y axis so we have a nice horizontal bar chart rather than a vertical one.

***#6:*** Reversing the legend so that the order of it matches up with the order of xG and xGA on the chart itself.

## Data Use Case 5: xG Assisted, Joining, and xG+xGA

The end result should look like this:



## Data Use Case 6: Heatmaps

## Data Use Case 6: Heatmaps

For this example we're going to do a defensive heatmap, looking at how often teams make a % of their overall defensive actions in certain zones, then comparing that % vs league average:

```
library(tidyverse)

heatmap = events %>%mutate(location.x = ifelse(location.x>120, 120, location.x),
location.y = ifelse(location.y>80, 80, location.y),
location.x = ifelse(location.x<0, 0, location.x),
location.y = ifelse(location.y<0, 0, location.y)) #1

heatmap$xbin <- cut(heatmap$location.x, breaks = seq(from=0, to=120, by = 20),include.lowest=TRUE )
heatmap$ybin <- cut(heatmap$location.y, breaks = seq(from=0, to=80, by = 20),include.lowest=TRUE) #2
```

***#1*** Some of the coordinates in our data sit outside the bounds of the pitch (you can see the layout of our pitch coordinates in our event spec, but it's 0-120 along the x axis and 0-80 along the y axis). This will cause issue with a heatmap and give you dodgy looking zones outside the pitch. So what we're doing here is using *ifelse()* to say 'if a location.x/y coordinate is outside the bounds that we want, then replace it with one that's within the boundaries. If it is not outside the bounds just leave it as is'.

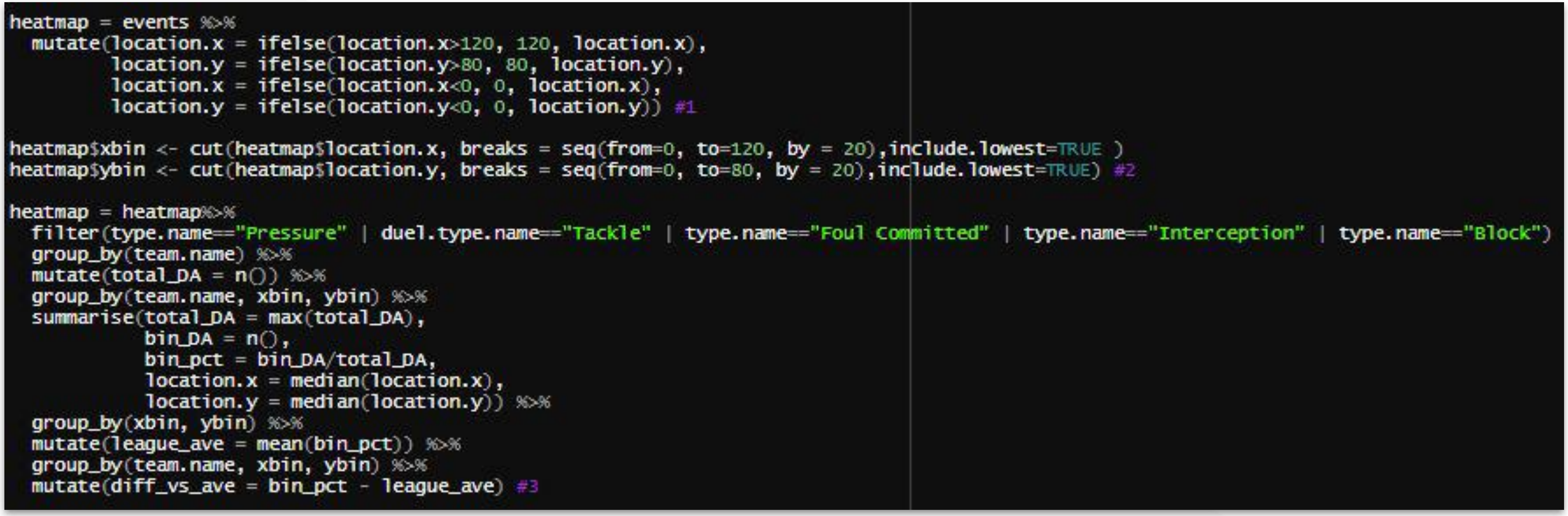
***#2*** *cut()* literally cuts up the data how you ask it to. Here, we're cutting along the x axis (from 0-120, again the length of our pitch according to our coordinates in the spec) and the y axis (0-80), and we're cutting them 'by' the value we feed it, in this case 20. So we're splitting it up into buckets of 20. This creates 6 buckets/zones along the x axis (120/20 = 6) and 4 along the y axis (80/20 = 4). This creates the buckets we need to plot our zones.

## Data Use Case 6: Heatmaps

```
heatmap = heatmap%>%
filter(type.name=="Pressure" | duel.type.name=="Tackle" |
type.name=="Foul Committed" | type.name=="Interception" |
type.name=="Block" ) %>%
group_by(team.name) %>%
mutate(total_DA = n()) %>%
group_by(team.name, xbin, ybin) %>%
summarise(total_DA = max(total_DA),
bin_DA = n(),
bin_pct = bin_DA/total_DA,
location.x = median(location.x),
location.y = median(location.y)) %>%
group_by(xbin, ybin) %>%
mutate(league_ave = mean(bin_pct)) %>%
group_by(team.name, xbin, ybin) %>%
mutate(diff_vs_ave = bin_pct - league_ave) #3
```

**#3:** This is using those buckets to create the zones. Let's break it down bit-by-bit:
- Filtering to only defensive events
- Grouping by team and getting how many defensive events they made in total ( *n()* just counts every row that you ask it to, so here we're counting every row for every team - i.e counting every defensive event for each team)
- Then we group again by team and the xbin/ybin to count how many defensive events a team has in a given bin/zone - that's what '*bin_DA = n()*' is doing. '*total_DA = max(total_DA),*' is just grabbing the team totals we made earlier. '*bin_pct = bin_DA/total_DA,*' is dividing the two to see what percentage of a team's overall defensive events were made in a given zone. The '*location.x = median(location.x/y)*' is doing what it says on the tin and getting the median coordinate for each zone. This is used later in the plotting.
- Then we ungroup and mutate to find the league average for each bin, followed by grouping by team/bin again subtracting the league average in each bin from each team's % in those bins to get the difference.

## Data Use Case 6: Heatmaps



```
heatmap = events %>%
  mutate(location.x = ifelse(location.x>120, 120, location.x),
         location.y = ifelse(location.y>80, 80, location.y),
         location.x = ifelse(location.x<0, 0, location.x),
         location.y = ifelse(location.y<0, 0, location.y)) #1

heatmap$xbin <- cut(heatmap$location.x, breaks = seq(from=0, to=120, by = 20),include.lowest=TRUE )
heatmap$ybin <- cut(heatmap$location.y, breaks = seq(from=0, to=80, by = 20),include.lowest=TRUE) #2

heatmap = heatmap%>%
  filter(type.name=="Pressure" | duel.type.name=="Tackle" | type.name=="Foul Committed" | type.name=="Interception" | type.name=="Block") %>%
  group_by(team.name) %>%
  mutate(total_DA = n()) %>%
  group_by(team.name, xbin, ybin) %>%
  summarise(total_DA = max(total_DA),
            bin_DA = n(),
            bin_pct = bin_DA/total_DA,
            location.x = median(location.x),
            location.y = median(location.y)) %>%
  group_by(xbin, ybin) %>%
  mutate(league_ave = mean(bin_pct)) %>%
  group_by(team.name, xbin, ybin) %>%
  mutate(diff_vs_ave = bin_pct - league_ave) #3
```

STATSBOMB

# Data Use Case 6: Heatmaps

Now onto the plotting. For this please install the package 'grid' if you do not have it, and load it in. You could use a package like 'ggsoccer' or 'SBPitch' for drawing the pitch, but for these purposes it's helpful to try and show you how to create your own pitch, should you want to:

```
library(grid)

defensiveactivitycolors <- c("#dc2429", "#dc2329", "#df272d", "#df3238", "#e14348", "#e44d51",
"#e35256", "#e76266", "#e9777b", "#ec8589", "#ec898d", "#ef9195",
"#ef9ea1", "#f0a6a9", "#f2abae", "#f4b9bc", "#f8d1d2", "#f9e0e2",
"#f7e1e3", "#f5e2e4", "#d4d5d8", "#d1d3d8", "#cdd2d6", "#c8cdd3", "#c0c7cd",
"#b9c0c8", "#b5bcc3", "#909ba5", "#8f9aa5", "#818c98", "#798590",
"#697785", "#526173", "#435367", "#3a4b60", "#2e4257", "#1d3048",
"#11263e", "#11273e", "#0d233a", "#020c16") #1
```

***#1:*** These are the colours we'll be using for our heatmap later on.

# Data Use Case 6: Heatmaps

```
ggplot(data= heatmap, aes(x = location.x, y = location.y, fill = diff_vs_ave, group =diff_vs_ave)) +
  geom_bin2d(binwidth = c(20, 20), position = "identity", alpha = 0.9) + #2
  annotate("rect",xmin = 0, xmax = 120, ymin = 0, ymax = 80, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 0, xmax = 60, ymin = 0, ymax = 80, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 18, xmax = 0, ymin = 18, ymax = 62, fill = NA, colour = "white", size = 0.6) +
  annotate("rect",xmin = 102, xmax = 120, ymin = 18, ymax = 62, fill = NA, colour = "white", size = 0.6) +
  annotate("rect",xmin = 0, xmax = 6, ymin = 30, ymax = 50, fill = NA, colour = "white", size = 0.6) +
  annotate("rect",xmin = 120, xmax = 114, ymin = 30, ymax = 50, fill = NA, colour = "white", size = 0.6) +
  annotate("rect",xmin = 120, xmax = 120.5, ymin =36, ymax = 44, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 0, xmax = -0.5, ymin =36, ymax = 44, fill = NA, colour = "black", size = 0.6) +
  annotate("segment", x = 60, xend = 60, y = -0.5, yend = 80.5, colour = "white", size = 0.6)+
  annotate("segment", x = 0, xend = 0, y = 0, yend = 80, colour = "black", size = 0.6)+
  annotate("segment", x = 120, xend = 120, y = 0, yend = 80, colour = "black", size = 0.6)+
  theme(rect = element_blank(),
     line = element_blank()) +
  annotate("point", x = 12 , y = 40, colour = "white", size = 1.05) +
  annotate("point", x = 108 , y = 40, colour = "white", size = 1.05) +
  annotate("path", colour = "white", size = 0.6,
       x=60+10*cos(seq(0,2*pi,length.out=2000)),
       y=40+10*sin(seq(0,2*pi,length.out=2000)))+
  annotate("point", x = 60 , y = 40, colour = "white", size = 1.05) +
  annotate("path", x=12+10*cos(seq(-0.3*pi,0.3*pi,length.out=30)), size = 0.6,
       y=40+10*sin(seq(-0.3*pi,0.3*pi,length.out=30)), col="white") +
  annotate("path", x=108-10*cos(seq(-0.3*pi,0.3*pi,length.out=30)), size = 0.6,
       y=40-10*sin(seq(-0.3*pi,0.3*pi,length.out=30)), col="white")  + #3
```

Bear in mind this next section of code--on this slide and the next--should be pasted into the console in one block. Be careful to do this when entering the code.

***#2:*** *'geom_bin2d'* is what will create the heatmap itself. We've set the binwidths to 20 as that's what we cut the pitch up into earlier along the x and y axis. Feeding 'div_vs_ave' to 'fill' and 'group' in the *ggplot()* will allow us to colour the heatmaps by that variable.

***#3:*** Everything up to here is what is drawing the pitch. There's a lot going on here and, rather than have it explained to you, just delete a line from it and see what disappears from the plot. Then you'll see which line is drawing the six-yard-box, which is drawing the goal etc.

## Data Use Case 6: Heatmaps

```
theme(axis.text.x=element_blank(),
      axis.title.x = element_blank(),
      axis.title.y = element_blank(),
      plot.caption=element_text(size=13,family="Source Sans Pro", hjust=0.5, vjust=0.5),
      plot.subtitle = element_text(size = 18, family="Source Sans Pro", hjust = 0.5),
      axis.text.y=element_blank(),
      legend.title = element_blank(),
      legend.text=element_text(size=22,family="Source Sans Pro"),
      legend.key.size = unit(1.5, "cm"),
      plot.title = element_text(margin = margin(r = 10, b = 10), face="bold",size = 32.5,
family="Source Sans Pro", colour = "black", hjust = 0.5),
      legend.direction = "vertical",
      axis.ticks=element_blank(),
      plot.background = element_rect(fill = "white"),
      strip.text.x = element_text(size=13,family="Source Sans Pro")) + #4
  scale_y_reverse() + #5
  scale_fill_gradientn(colours = defensiveactivitycolors, trans = "reverse", labels =
scales::percent_format(accuracy = 1), limits = c(0.03, -0.03)) + #6
  labs(title = "Where Do Teams Defend vs League Average?", subtitle = "FA Women's Super
League, 2020/21") + #7
  coord_fixed(ratio = 95/100) + #8
```

***#4:*** Again more themeing. You can change this to be whatever you like to fit your aesthetic preferences.

***#5:*** Reversing the y axis so the pitch is the correct way round along that axis (0 is left in SBD coordinates, but starts out as right in ggplot).

***#6:*** Here we're setting the parameters for the fill colouring of heatmaps. First we're feeding the 'defensiveactivitycolors' we set earlier into the '*colours*' parameter, '*trans = "reverse"*' is there to reverse the output so red = high. '*labels = scales::percent_format(accuracy = 1)*' formats the text on the legend as a percentage rather than a raw number and '*limits = c(0.03, -0.03)*' sets the limits of the chart to 3%/-3% (reversed because of the previous trans = reverse).

***#7:*** Setting the title and subtitle of the chart.

***#8:*** '*coord_fixed()*' allows us to set the aspect ratio of the chart to our liking. Means the chart doesn't come out looking all stretched along one of the axes.

# Data Use Case 6: Heatmaps

```
annotation_custom(grob = linesGrob(arrow=arrow(type="open", ends="last",
                                   length=unit(2.55,"mm")), gp=gpar(col="black", fill=NA, lwd=2.2)),
            xmin=25, xmax = 95, ymin = -83, ymax = -83) + #9
facet_wrap(~team.name)+ #10
guides(fill = guide_legend(reverse = TRUE)) #11
```

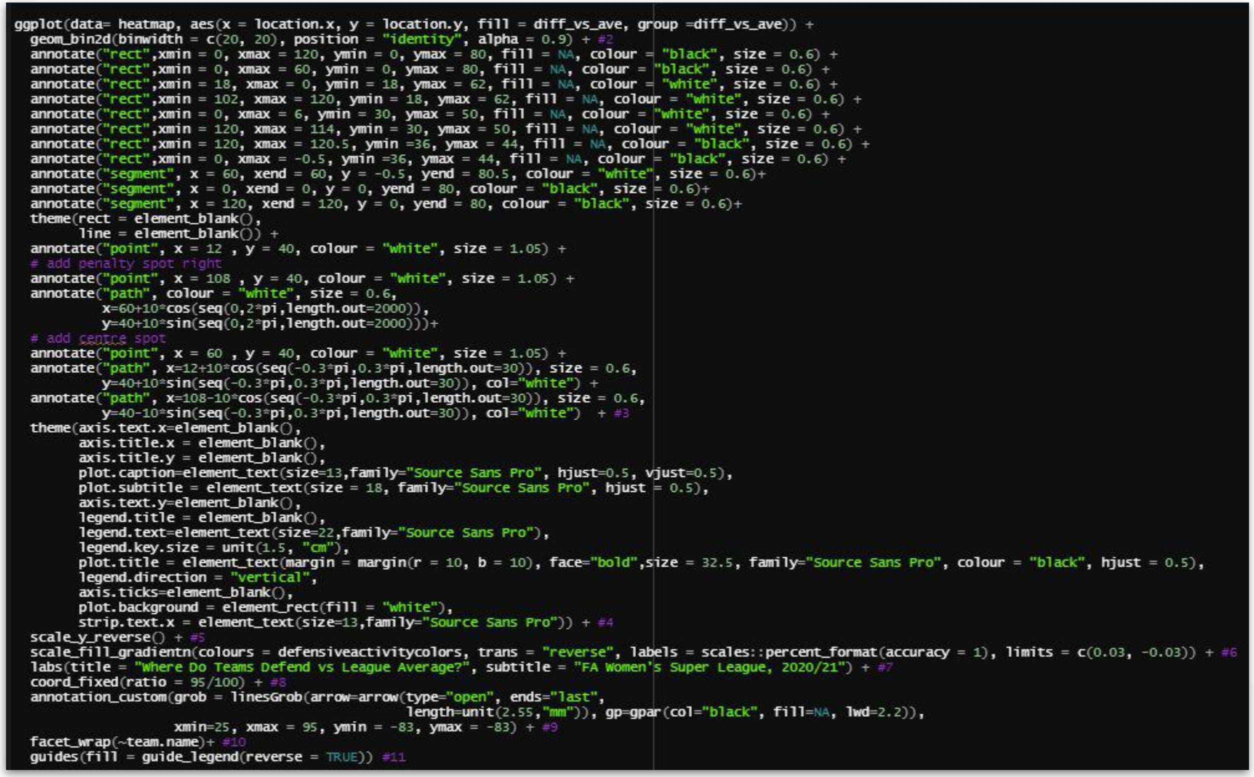
***#9:*** This is what the grid package is used for. It's drawing the arrow across the pitches to indicate direction of play. There's multiple ways you could accomplish though, up to you how you do it.

***#10:*** '*facet_wrap()*' creates separate 'facets' for your chart according to the variable you give it. Without it, we'd just be plotting every team's numbers all at once on chart. With it, we get every team on their own individual pitch.

***#11:*** Our previous *trans* = *reverse* also reverses the legend, so to get it back with the positive numbers pointing upwards we can re-reverse it.

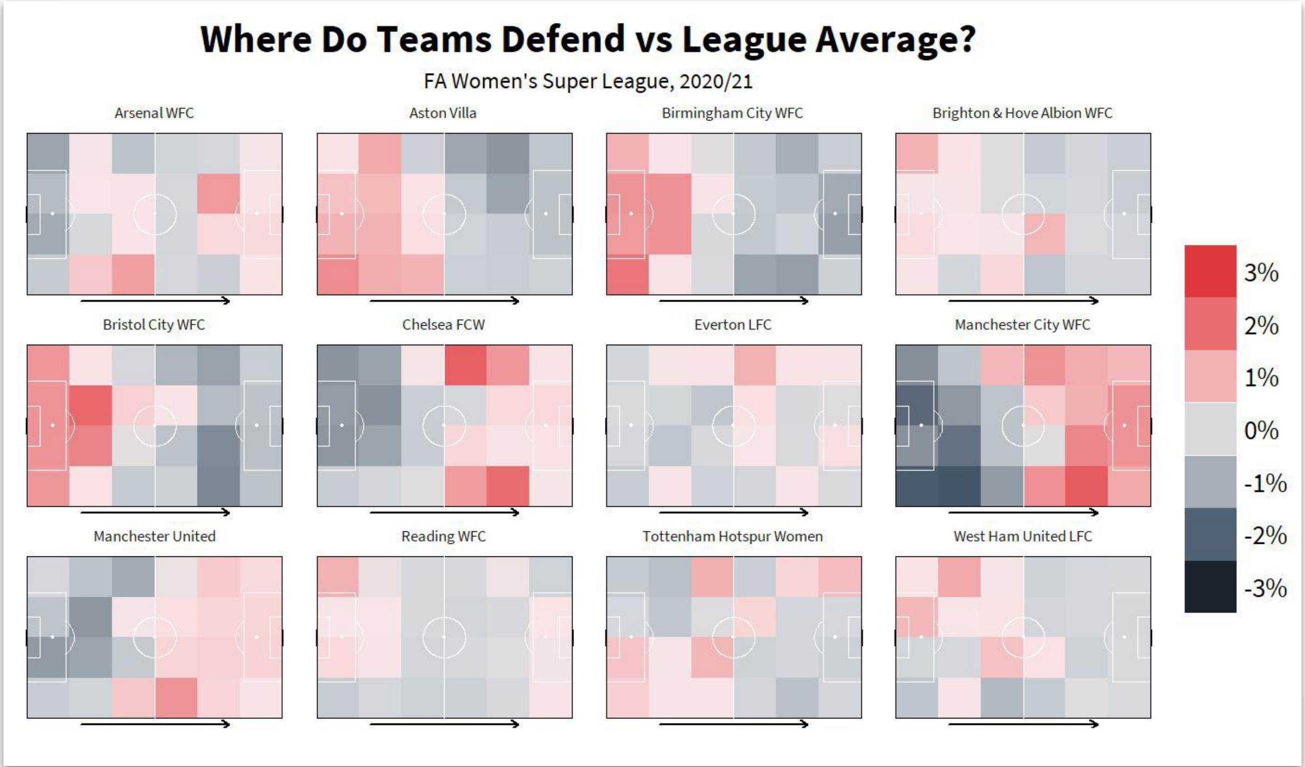
STATSBOMB

## Data Use Case 6: Heatmaps



```r
ggplot(data= heatmap, aes(x = location.x, y = location.y, fill = diff_vs_ave, group =diff_vs_ave)) +
  geom_bin2d(binwidth = c(20, 20), position = "identity", alpha = 0.9) + #2
  annotate("rect",xmin = 0, xmax = 120, ymin = 0, ymax = 80, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 0, xmax = 60, ymin = 0, ymax = 80, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 18, xmax = 0, ymin = 18, ymax = 62, fill = NA, colour = "white", size = 0.6) +
  annotate("rect",xmin = 102, xmax = 120, ymin = 18, ymax = 62, fill = NA, colour = "white", size = 0.6) +
  annotate("rect",xmin = 0, xmax = 6, ymin = 30, ymax = 50, fill = NA, colour = "white", size = 0.6) +
  annotate("rect",xmin = 120, xmax = 114, ymin = 30, ymax = 50, fill = NA, colour = "white", size = 0.6) +
  annotate("rect",xmin = 120, xmax = 120.5, ymin =36, ymax = 44, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 0, xmax = -0.5, ymin =36, ymax = 44, fill = NA, colour = "black", size = 0.6) +
  annotate("segment", x = 60, xend = 60, y = -0.5, yend = 80.5, colour = "white", size = 0.6)+
  annotate("segment", x = 0, xend = 0, y = 0, yend = 80, colour = "black", size = 0.6)+
  annotate("segment", x = 120, xend = 120, y = 0, yend = 80, colour = "black", size = 0.6)+
  theme(rect = element_blank(),
        line = element_blank()) +
  annotate("point", x = 12 , y = 40, colour = "white", size = 1.05) +
  # add penalty spot right
  annotate("point", x = 108 , y = 40, colour = "white", size = 1.05) +
  annotate("path", colour = "white", size = 0.6,
           x=60+10*cos(seq(0,2*pi,length.out=2000)),
           y=40+10*sin(seq(0,2*pi,length.out=2000)))+
  # add centre spot
  annotate("point", x = 60 , y = 40, colour = "white", size = 1.05) +
  annotate("path", x=12+10*cos(seq(-0.3*pi,0.3*pi,length.out=30)), size = 0.6,
           y=40+10*sin(seq(-0.3*pi,0.3*pi,length.out=30)), col="white") +
  annotate("path", x=108-10*cos(seq(-0.3*pi,0.3*pi,length.out=30)), size = 0.6,
           y=40-10*sin(seq(-0.3*pi,0.3*pi,length.out=30)), col="white")  + #3
  theme(axis.text.x=element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        plot.caption=element_text(size=13,family="Source Sans Pro", hjust=0.5, vjust=0.5),
        plot.subtitle = element_text(size = 18, family="Source Sans Pro", hjust = 0.5),
        axis.text.y=element_blank(),
        legend.title = element_blank(),
        legend.text=element_text(size=22,family="Source Sans Pro"),
        legend.key.size = unit(1.5, "cm"),
        plot.title = element_text(margin = margin(r = 10, b = 10), face="bold",size = 32.5, family="Source Sans Pro", colour = "black", hjust = 0.5),
        legend.direction = "vertical",
        axis.ticks=element_blank(),
        plot.background = element_rect(fill = "white"),
        strip.text.x = element_text(size=13,family="Source Sans Pro")) + #4
  scale_y_reverse() + #5
  scale_fill_gradientn(colours = defensiveactivitycolors, trans = "reverse", labels = scales::percent_format(accuracy = 1), limits = c(0.03, -0.03)) + #6
  labs(title = "Where Do Teams Defend vs League Average?", subtitle = "FA Women's Super League, 2020/21") + #7
  coord_fixed(ratio = 95/100) + #8
  annotation_custom(grob = linesGrob(arrow=arrow(type="open", ends="last",
                                                  length=unit(2.55,"mm")), gp=gpar(col="black", fill=NA, lwd=2.2)),
                    xmin=25, xmax = 95, ymin = -83, ymax = -83) + #9
  facet_wrap(~team.name)+ #10
  guides(fill = guide_legend(reverse = TRUE)) #11
```

# Data Use Case 6: Heatmaps



## Data Use Case 7: Shot Maps

STATSBOMB

## Data Use Case 7: Shot Maps

Another of the quintessential football visualisations, shot maps come in many shapes and sizes with an inconsistent overlap in design language between them. This version will attempt to give you the basics, let you get to grip with how to put one of these together so that if you want to elaborate or make any of your own changes you can explore outwards from it. Be forewarned though - the options for what makes a good, readable shot map are surprisingly small when you get into visualising it!

```
shots = events %>%
  filter(type.name=="Shot" & (shot.type.name!="Penalty" | is.na(shot.type.name)) & player.name=="Samantha May Kerr") #1

shotmapxgcolors <- c("#192780", "#2a5d9f", "#40a7d0", "#87cdcf", "#e7f8e6", "#f4ef95", "#FDE960", "#FCDC5F",
            "#F5B94D", "#F0983E", "#ED8A37", "#E66424", "#D54F1B", "#DC2608", "#BF0000", "#7F0000", "#5F0000") #2
```

***#1:*** Simple filtering, leaving out penalties. Choose any player you like of course.

***#2:*** Much like the defensive activity colours earlier, these will set the colours for our xG values.

# Data Use Case 7: Shot Maps

```
ggplot() +
 annotate("rect",xmin = 0, xmax = 120, ymin = 0, ymax = 80, fill = NA, colour = "black", size = 0.6) +
 annotate("rect",xmin = 0, xmax = 60, ymin = 0, ymax = 80, fill = NA, colour = "black", size = 0.6) +
 annotate("rect",xmin = 18, xmax = 0, ymin = 18, ymax = 62, fill = NA, colour = "black", size = 0.6) +
 annotate("rect",xmin = 102, xmax = 120, ymin = 18, ymax = 62, fill = NA, colour = "black", size = 0.6) +
 annotate("rect",xmin = 0, xmax = 6, ymin = 30, ymax = 50, fill = NA, colour = "black", size = 0.6) +
 annotate("rect",xmin = 120, xmax = 114, ymin = 30, ymax = 50, fill = NA, colour = "black", size = 0.6) +
 annotate("rect",xmin = 120, xmax = 120.5, ymin =36, ymax = 44, fill = NA, colour = "black", size = 0.6) +
 annotate("rect",xmin = 0, xmax = -0.5, ymin =36, ymax = 44, fill = NA, colour = "black", size = 0.6) +
 annotate("segment", x = 60, xend = 60, y = -0.5, yend = 80.5, colour = "black", size = 0.6)+
 annotate("segment", x = 0, xend = 0, y = 0, yend = 80, colour = "black", size = 0.6)+
 annotate("segment", x = 120, xend = 120, y = 0, yend = 80, colour = "black", size = 0.6)+
 theme(rect = element_blank(),
    line = element_blank()) +
 # add penalty spot right
 annotate("point", x = 108 , y = 40, colour = "black", size = 1.05) +
 annotate("path", colour = "black", size = 0.6,
      x=60+10*cos(seq(0,2*pi,length.out=2000)),
      y=40+10*sin(seq(0,2*pi,length.out=2000)))+
 # add centre spot
 annotate("point", x = 60 , y = 40, colour = "black", size = 1.05) +
 annotate("path", x=12+10*cos(seq(-0.3*pi,0.3*pi,length.out=30)), size = 0.6,
      y=40+10*sin(seq(-0.3*pi,0.3*pi,length.out=30)), col="black") +
 annotate("path", x=107.84-10*cos(seq(-0.3*pi,0.3*pi,length.out=30)), size = 0.6,
      y=40-10*sin(seq(-0.3*pi,0.3*pi,length.out=30)), col="black") +
 geom_point(data = shots, aes(x = location.x, y = location.y, fill = shot.statsbomb_xg, shape = shot.body_part.name),
       size = 6, alpha = 0.8) + #3
```

Again bear in mind that this next set of ggplot code (on this slide and the next two) should be pasted in one block.

***#3:*** Here's where the actual plotting of shots comes in, via *geom_point*. We're using the the xG values as the *fill* and the body part for the *shape* of the points. This could reasonably be anything though. You could even add in *colour* parameters which would change the colour of the outline of the shape.

# Data Use Case 7: Shot Maps

```
 theme(axis.text.x=element_blank(),
      axis.title.x = element_blank(),
      axis.title.y = element_blank(),
      plot.caption=element_text(size=13,family="Source Sans Pro", hjust=0.5, vjust=0.5),
      plot.subtitle = element_text(size = 18, family="Source Sans Pro", hjust = 0.5),
      axis.text.y=element_blank(),
      legend.position = "top",
      legend.title=element_text(size=22,family="Source Sans Pro"),
      legend.text=element_text(size=20,family="Source Sans Pro"),
      legend.margin = margin(c(20, 10, -85, 50)),
      legend.key.size = unit(1.5, "cm"),
      plot.title = element_text(margin = margin(r = 10, b = 10), face="bold",size = 32.5, family="Source Sans
Pro", colour = "black", hjust = 0.5),
      legend.direction = "horizontal",
      axis.ticks=element_blank(),
      aspect.ratio = c(65/100),
      plot.background = element_rect(fill = "white"),
      strip.text.x = element_text(size=13,family="Source Sans Pro")) +
 labs(title = "Sam Kerr, Shot Map", subtitle = "FA Women's Super League, 2020/21") + #4
 scale_fill_gradientn(colours = shotmapxgcolors, limit = c(0,0.8), oob=scales::squish, name = "Expected Goals
Value") + #5
```

***#4:*** Again titling. This can be done dynamically so that it changes according to the player/season etc but we will leave that for now. Feel free to explore for youself though.

***#5:*** Same as last time but worth pointing out that '*name*' allows you to change the title of a legend from within the gradient setting.

# Data Use Case 7: Shot Maps

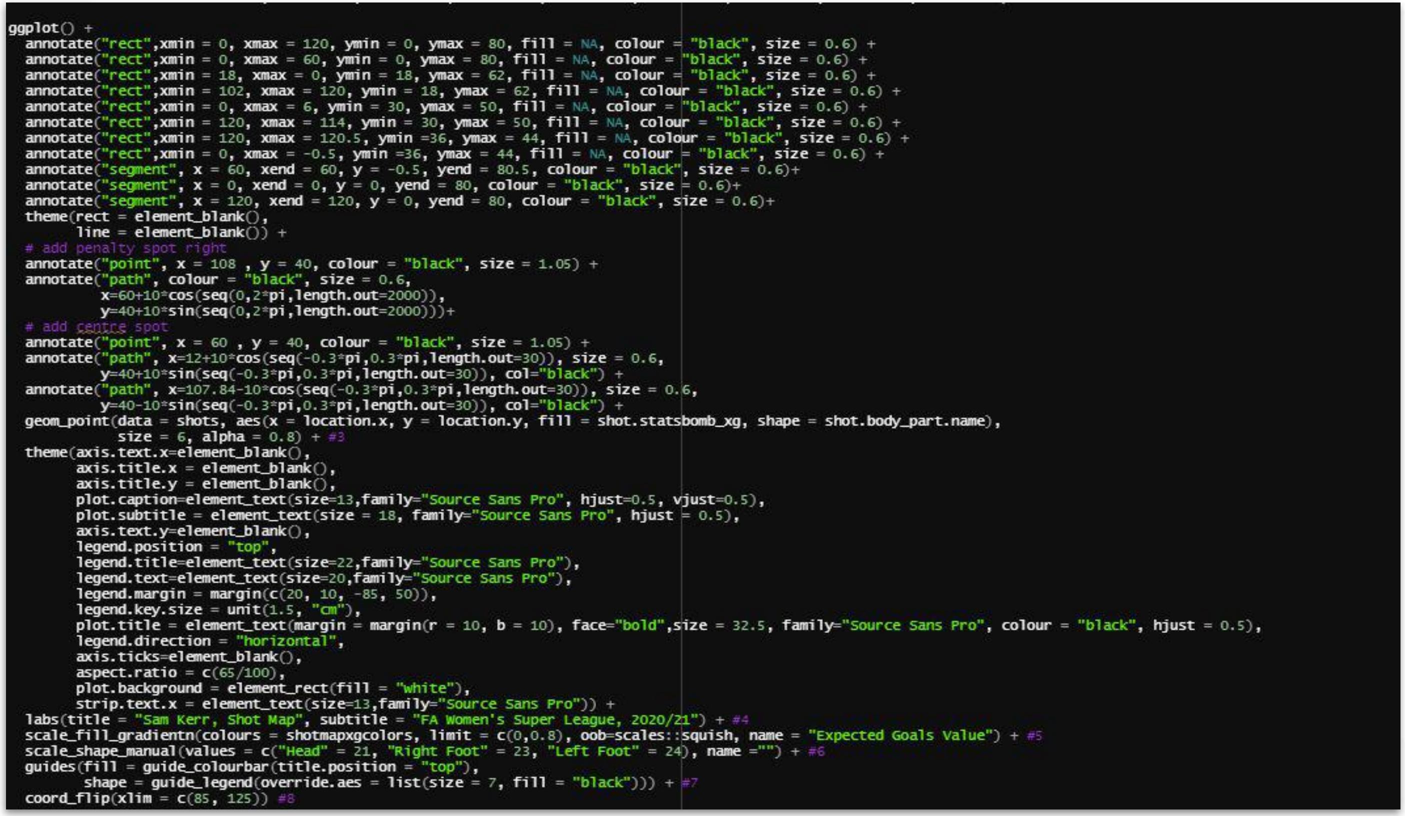
```
scale_shape_manual(values = c("Head" = 21, "Right Foot" = 23, "Left Foot" = 24), name ="") + #6
 guides(fill = guide_colourbar(title.position = "top"),
       shape = guide_legend(override.aes = list(size = 7, fill = "black"))) + #7
 coord_flip(xlim = c(85, 125)) #8
```

***#6:*** Setting the shapes for each body part name. The shape numbers correspond to ggplot's pre-set shapes, which you can find here. The shapes numbered 21 and up are the ones which have inner colouring (controlled by *fill*) and outline colouring (controlled by *colour*) so that's why those have been chosen here. *oob=scales::squish* takes any values that are outside the bounds of our limits and squishes them within them.

***#7:*** *guides()* allows you to alter the legends for shape, fill and so on. Here we are changing the the title position for the *fill* so that it is positioned above the legend, as well as changing the size and colour of the shape symbols on that legend.

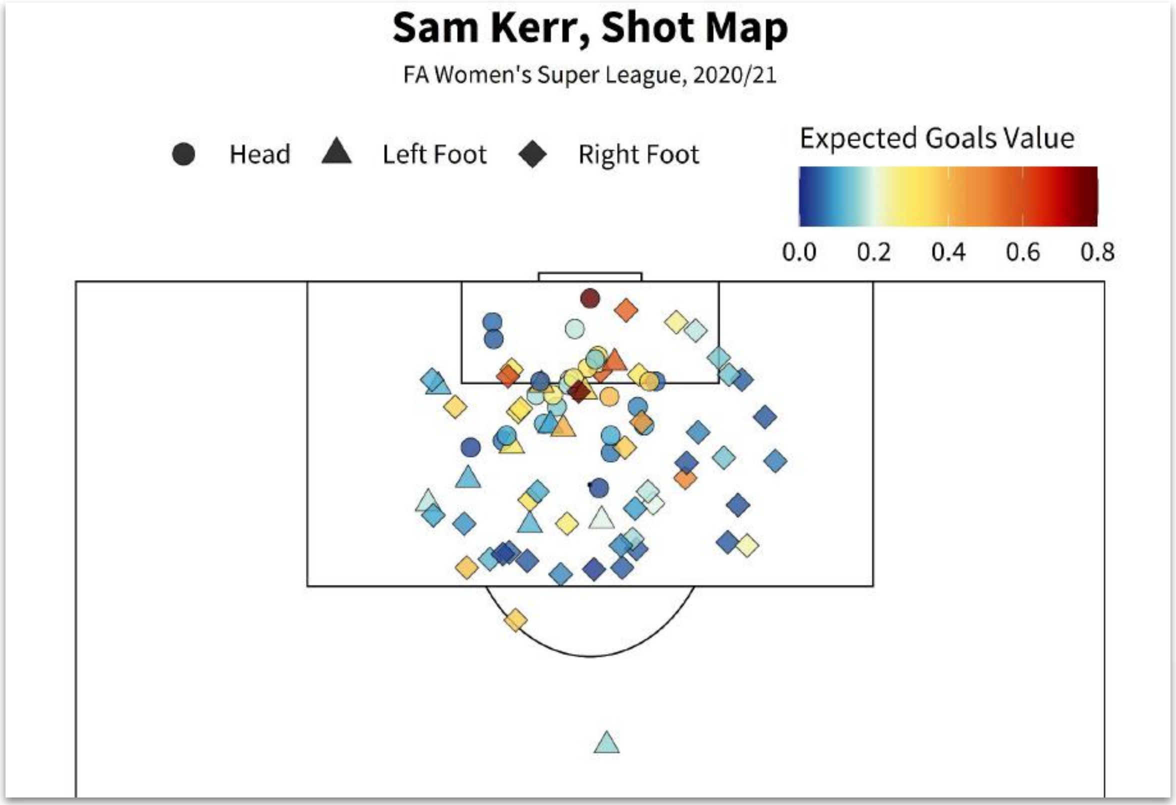
***#8:*** *coord_flip()* does what it says on the tin - switches the x and y axes. *xlim* allows us to set boundaries for the x axis so that we can show only a certain part of the pitch, giving us:

# Data Use Case 7: Shot Maps



```
ggplot() +
  annotate("rect",xmin = 0, xmax = 120, ymin = 0, ymax = 80, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 0, xmax = 60, ymin = 0, ymax = 80, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 18, xmax = 0, ymin = 18, ymax = 62, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 102, xmax = 120, ymin = 18, ymax = 62, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 0, xmax = 6, ymin = 30, ymax = 50, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 120, xmax = 114, ymin = 30, ymax = 50, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 120, xmax = 120.5, ymin =36, ymax = 44, fill = NA, colour = "black", size = 0.6) +
  annotate("rect",xmin = 0, xmax = -0.5, ymin =36, ymax = 44, fill = NA, colour = "black", size = 0.6) +
  annotate("segment", x = 60, xend = 60, y = -0.5, yend = 80.5, colour = "black", size = 0.6)+
  annotate("segment", x = 0, xend = 0, y = 0, yend = 80, colour = "black", size = 0.6)+
  annotate("segment", x = 120, xend = 120, y = 0, yend = 80, colour = "black", size = 0.6)+
  theme(rect = element_blank(),
        line = element_blank()) +
  # add penalty spot right
  annotate("point", x = 108 , y = 40, colour = "black", size = 1.05) +
  annotate("path", colour = "black", size = 0.6,
           x=60+10*cos(seq(0,2*pi,length.out=2000)),
           y=40+10*sin(seq(0,2*pi,length.out=2000)))+
  # add centre spot
  annotate("point", x = 60 , y = 40, colour = "black", size = 1.05) +
  annotate("path", x=12+10*cos(seq(-0.3*pi,0.3*pi,length.out=30)), size = 0.6,
           y=40+10*sin(seq(-0.3*pi,0.3*pi,length.out=30)), col="black") +
  annotate("path", x=107.84-10*cos(seq(-0.3*pi,0.3*pi,length.out=30)), size = 0.6,
           y=40-10*sin(seq(-0.3*pi,0.3*pi,length.out=30)), col="black") +
  geom_point(data = shots, aes(x = location.x, y = location.y, fill = shot.statsbomb_xg, shape = shot.body_part.name),
              size = 6, alpha = 0.8) + #3
  theme(axis.text.x=element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        plot.caption=element_text(size=13,family="Source Sans Pro", hjust=0.5, vjust=0.5),
        plot.subtitle = element_text(size = 18, family="Source Sans Pro", hjust = 0.5),
        axis.text.y=element_blank(),
        legend.position = "top",
        legend.title=element_text(size=22,family="Source Sans Pro"),
        legend.text=element_text(size=20,family="Source Sans Pro"),
        legend.margin = margin(c(20, 10, -85, 50)),
        legend.key.size = unit(1.5, "cm"),
        plot.title = element_text(margin = margin(r = 10, b = 10), face="bold",size = 32.5, family="Source Sans Pro", colour = "black", hjust = 0.5),
        legend.direction = "horizontal",
        axis.ticks=element_blank(),
        aspect.ratio = c(65/100),
        plot.background = element_rect(fill = "white"),
        strip.text.x = element_text(size=13,family="Source Sans Pro")) +
  labs(title = "Sam Kerr, Shot Map", subtitle = "FA Women's Super League, 2020/21") + #4
  scale_fill_gradientn(colours = shotmapxgcolors, limit = c(0,0.8), oob=scales::squish, name = "Expected Goals Value") + #5
  scale_shape_manual(values = c("Head" = 21, "Right Foot" = 23, "Left Foot" = 24), name ="") + #6
  guides(fill = guide_colourbar(title.position = "top"),
         shape = guide_legend(override.aes = list(size = 7, fill = "black"))) + #7
  coord_flip(xlim = c(85, 125)) #8
```

# Data Use Case 7: Shot Maps



## We hope You Enjoy the Data!

Any questions please email Euan Dewar, Senior Analyst

**euan.dewar@statsbomb.com**

STATSBOMB