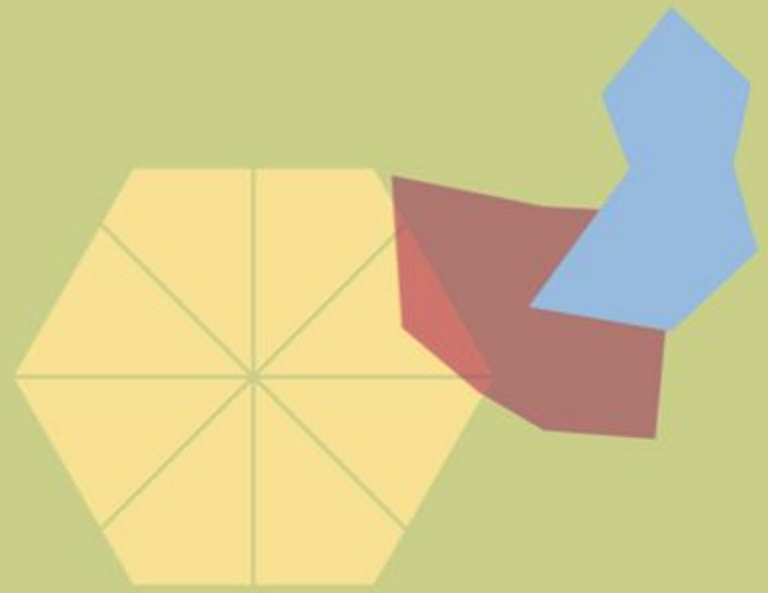
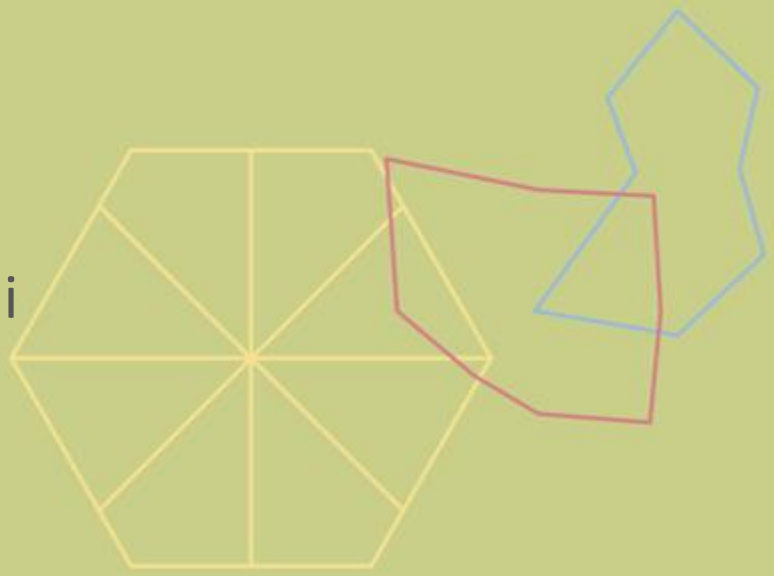
OXFORD UNIVERSITY STATISTICAL CONSULTING



# Graphics in R

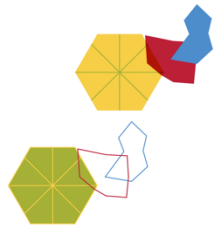
Maria Christodoulou and Mariagrazia Zottoli



# Graphics in Base R

# Graphics

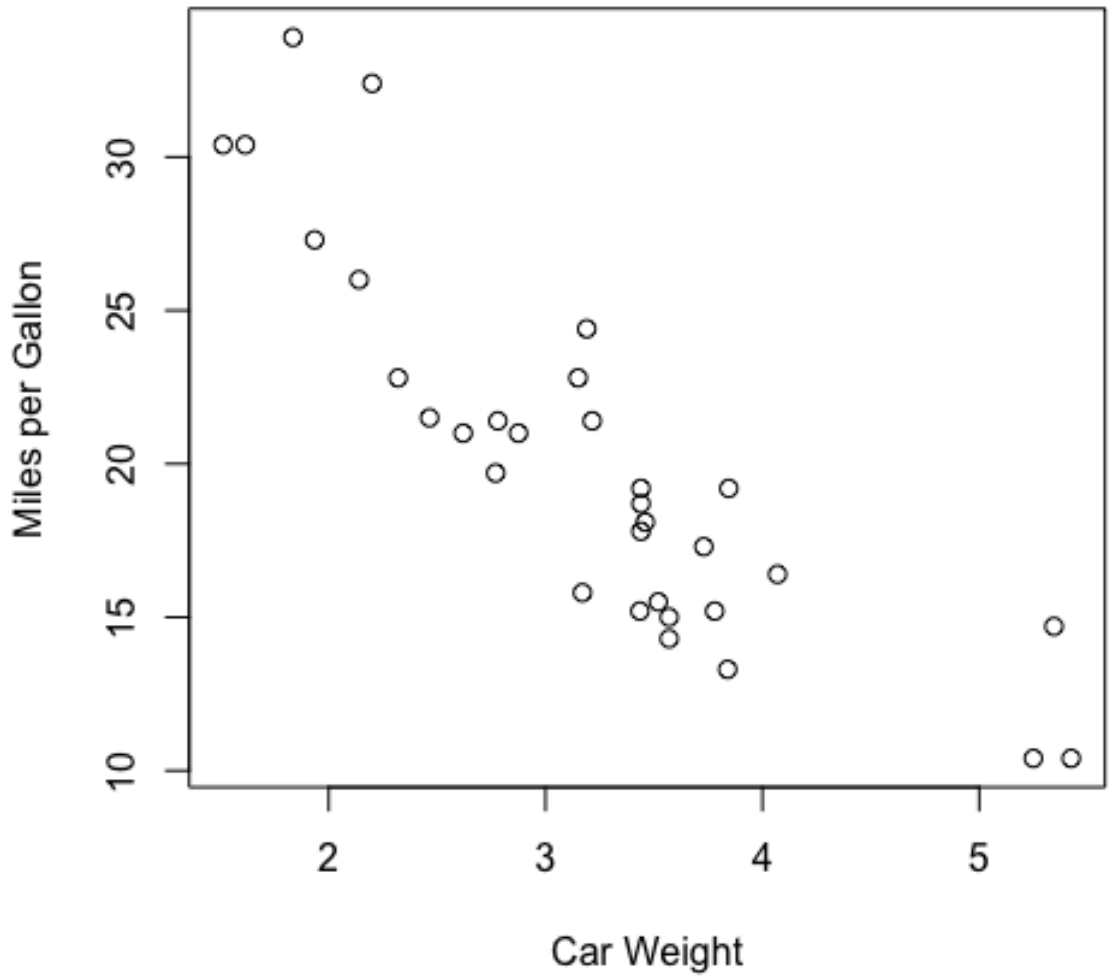
- You can use R to create publication quality graphics
- There are excellent specialised graphics packages available such as `ggplot2`
- You can use graphics to produce anything from scatterplots, to phylogenies, and distribution maps



# Graphics - Scatterplots

- The `plot()` function creates an x-y scatterplot

```
plot(mtcars$wt,
  mtcars$mpg,
xlab="Car Weight",
ylab="Miles per Gallon")
```

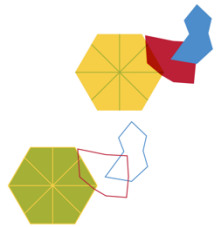


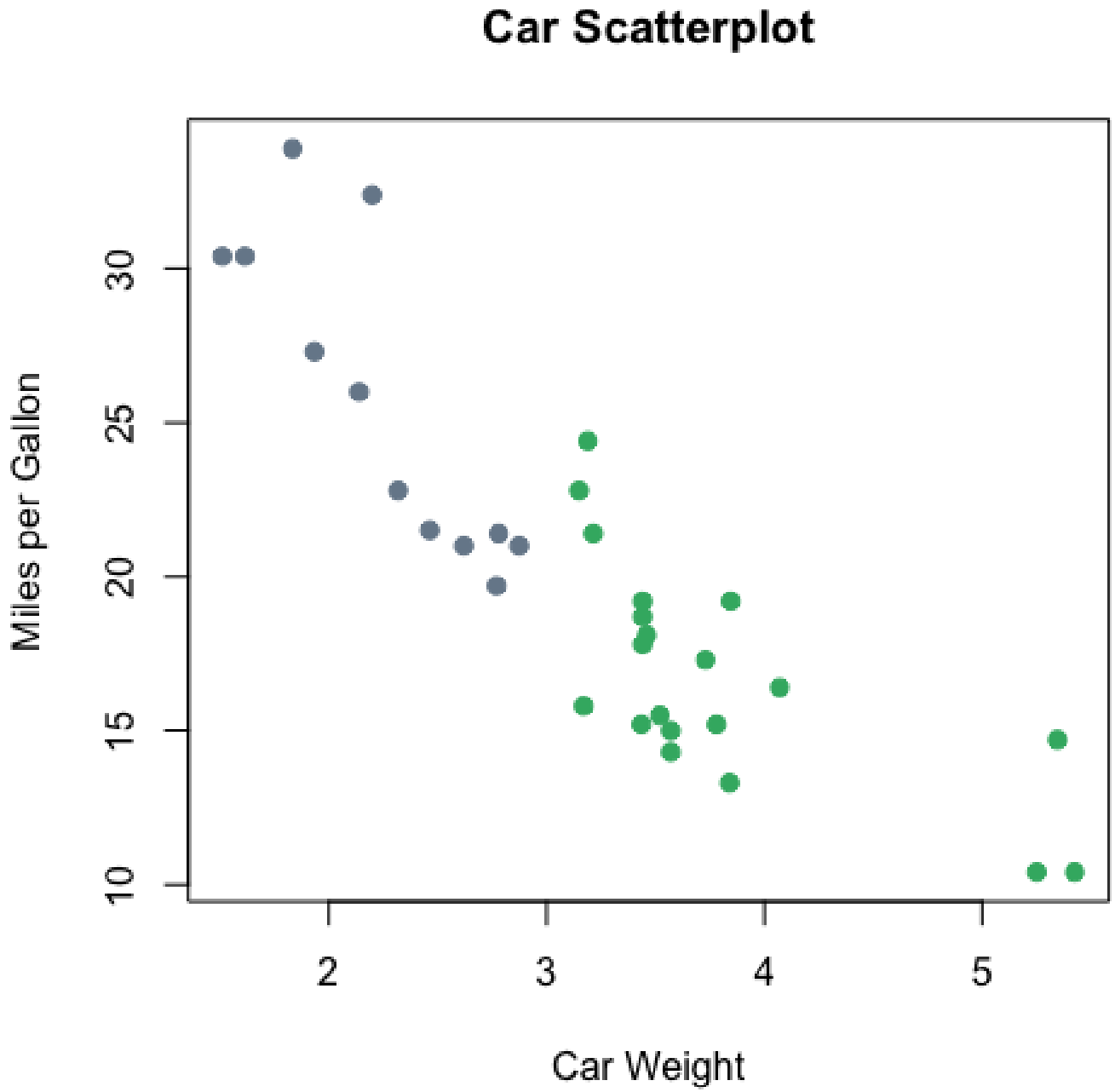
# Graphics - Scatterplots

- Change the look of your graphs using par (type `?par` in the console for more info)

```
> plot(mtcars$wt, mtcars$mpg, xlab="Car Weight", ylab="Miles per Gallon",main="Car Scatterplot",
  pch=19, col= ifelse(mtcars$wt>0 & mtcars$wt<=3, "lightslategray","mediumseagreen"))
```

- Google “colors in R” if you need some colour inspiration

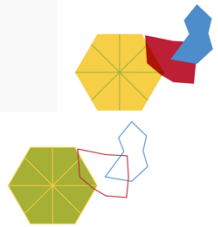


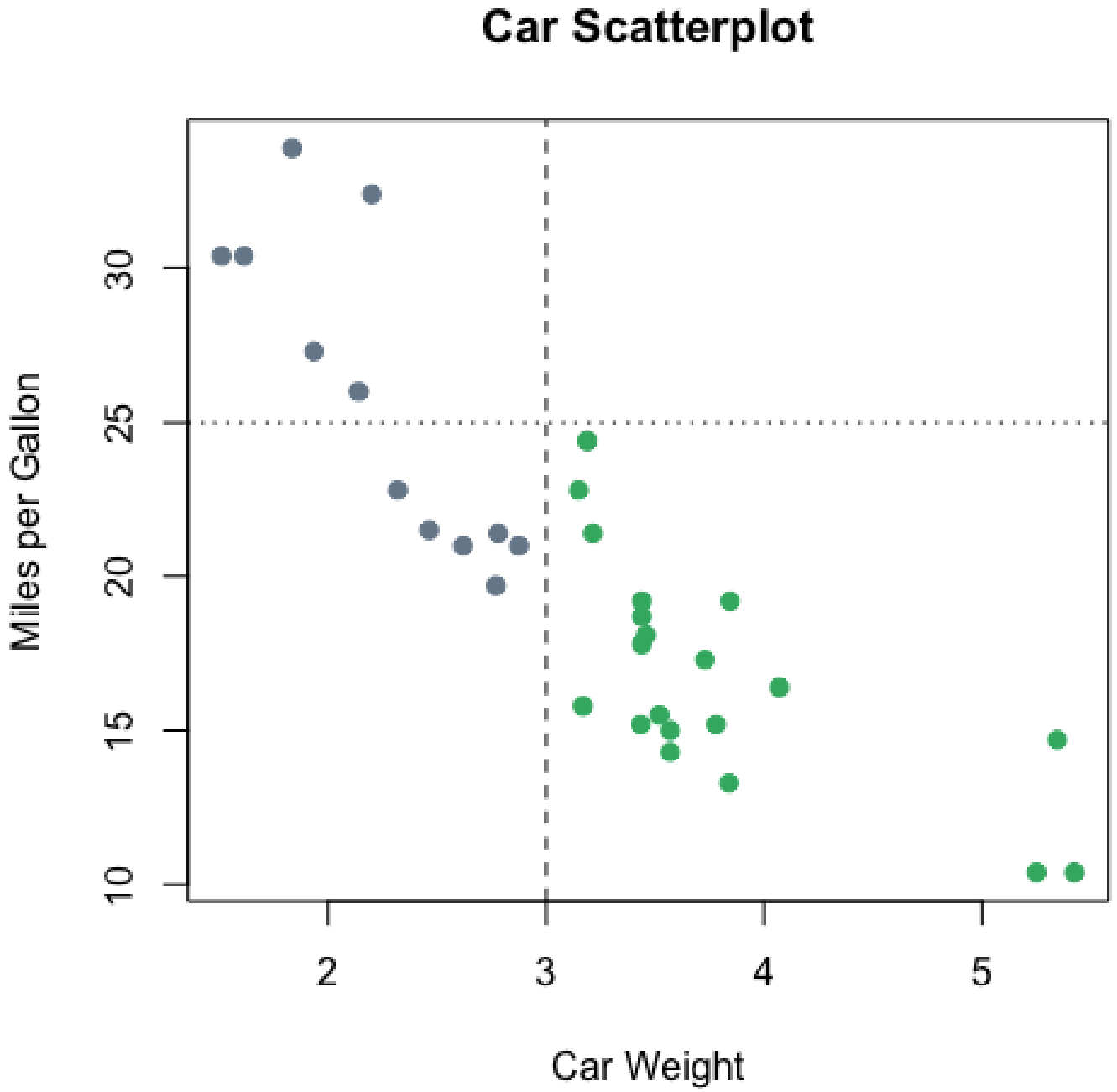

Car Scatterplot
Miles per Gallon
10
15
20
25
30
2
3
4
5
Car Weight

# Graphics - Scatterplots

You can add lines to your scatterplots using the `abline()` function

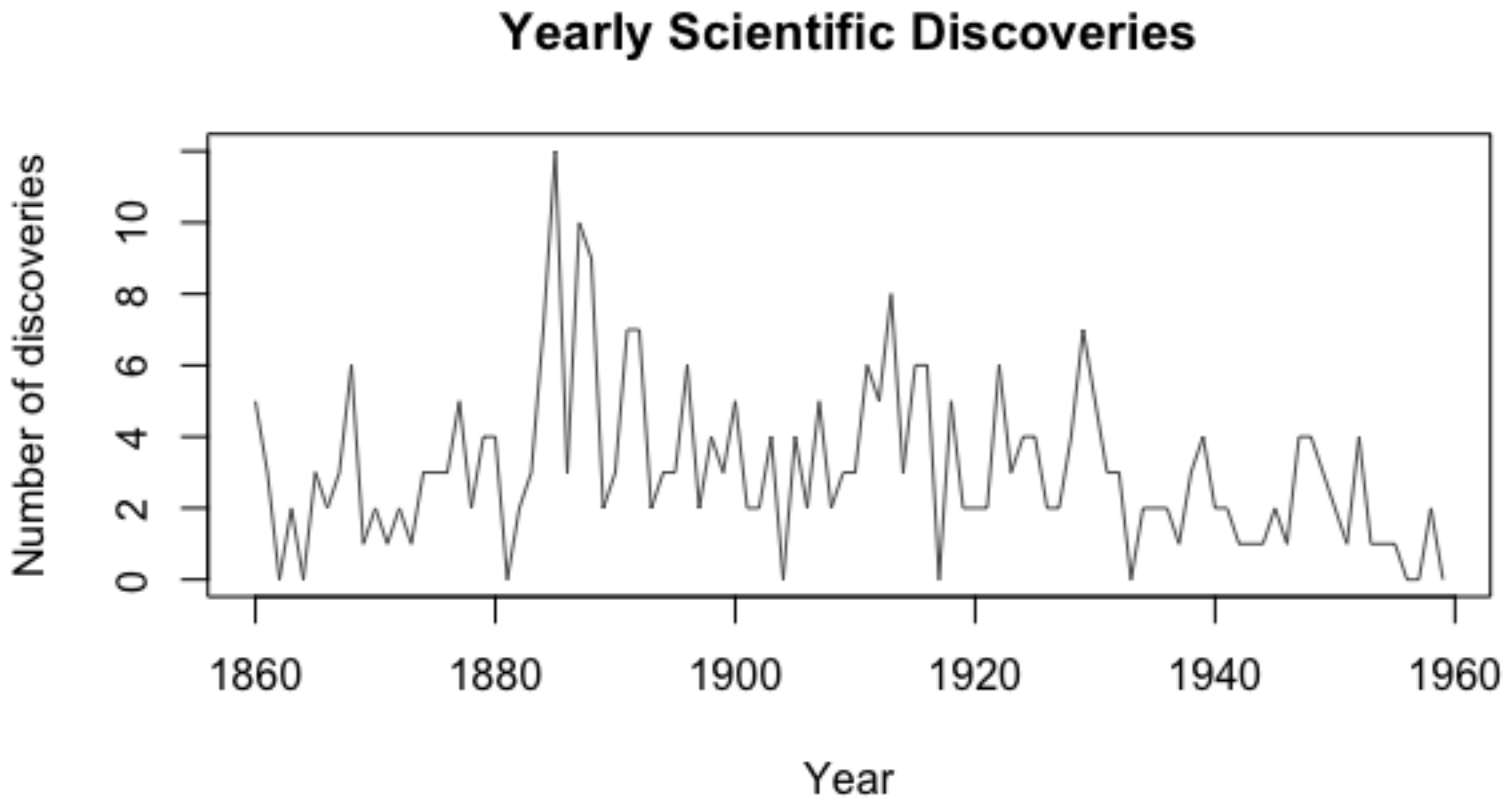
```
plot(mtcars$wt, mtcars$mpg, xlab="Car Weight", ylab="Miles per Gallon",
     main="Car Scatterplot", pch=19, col= ifelse(mtcars$wt>0 & mtcars$wt<=3,
                                                 "lightslategray","mediumseagreen"))

abline(v=3, colour="grey", lty="dashed")

abline(h=25, colour="black", lty="dotted")
```

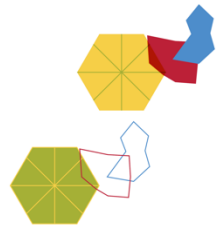



Car Scatterplot
Miles per Gallon
10
15
20
25
30
2
3
4
5
Car Weight

# Graphics – Line plots

```
> plot(seq(1860,1959,1),discoveries, xlab="Year", ylab="Number of discoveries", type="l",
main="Yearly Scientific Discoveries", col="gray35")
```



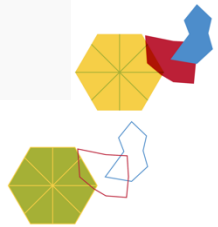



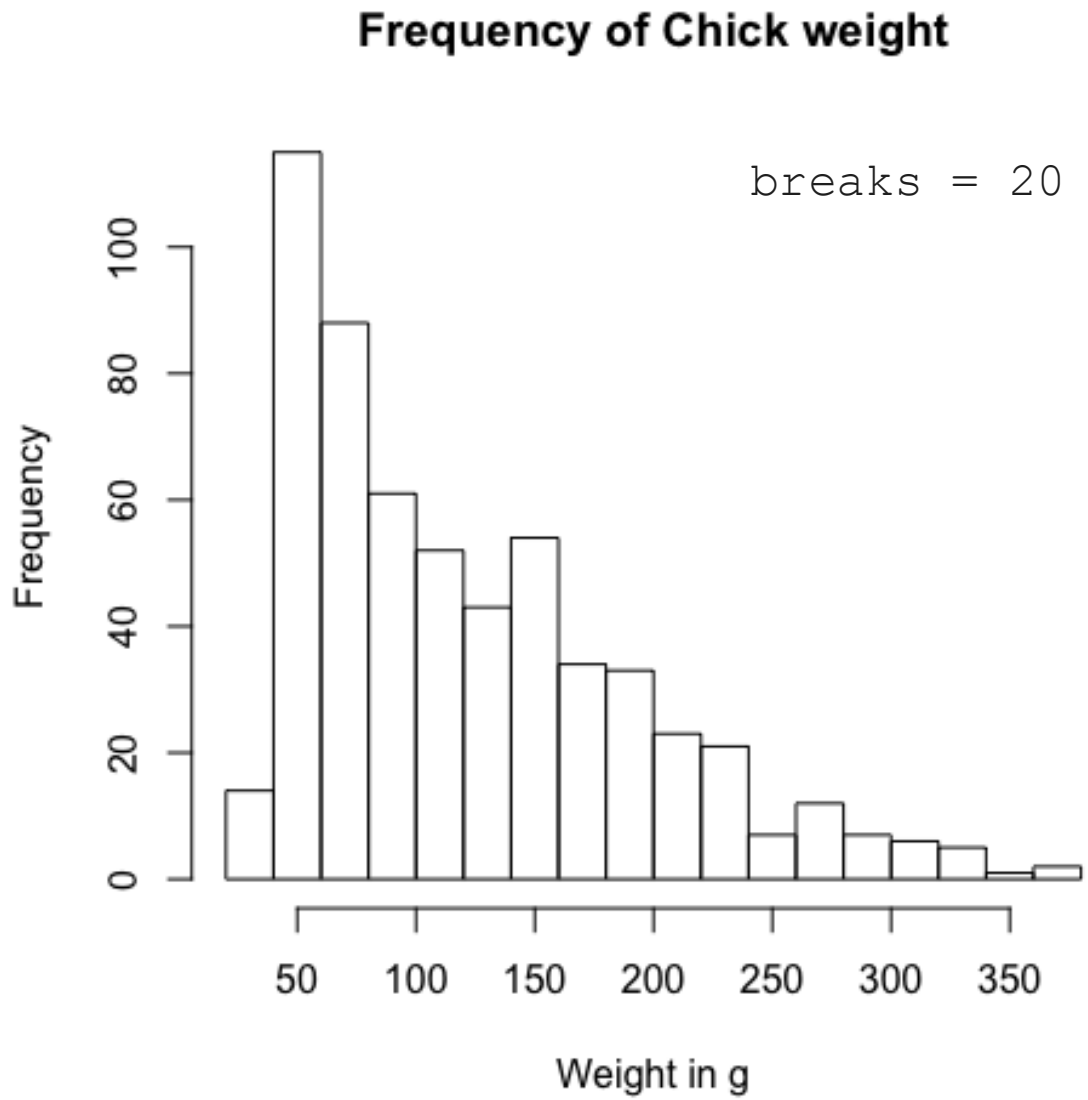
# Graphics - Histograms

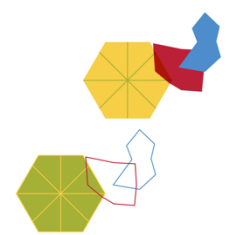
- Histograms are used for continuous data that have been grouped in “bins”

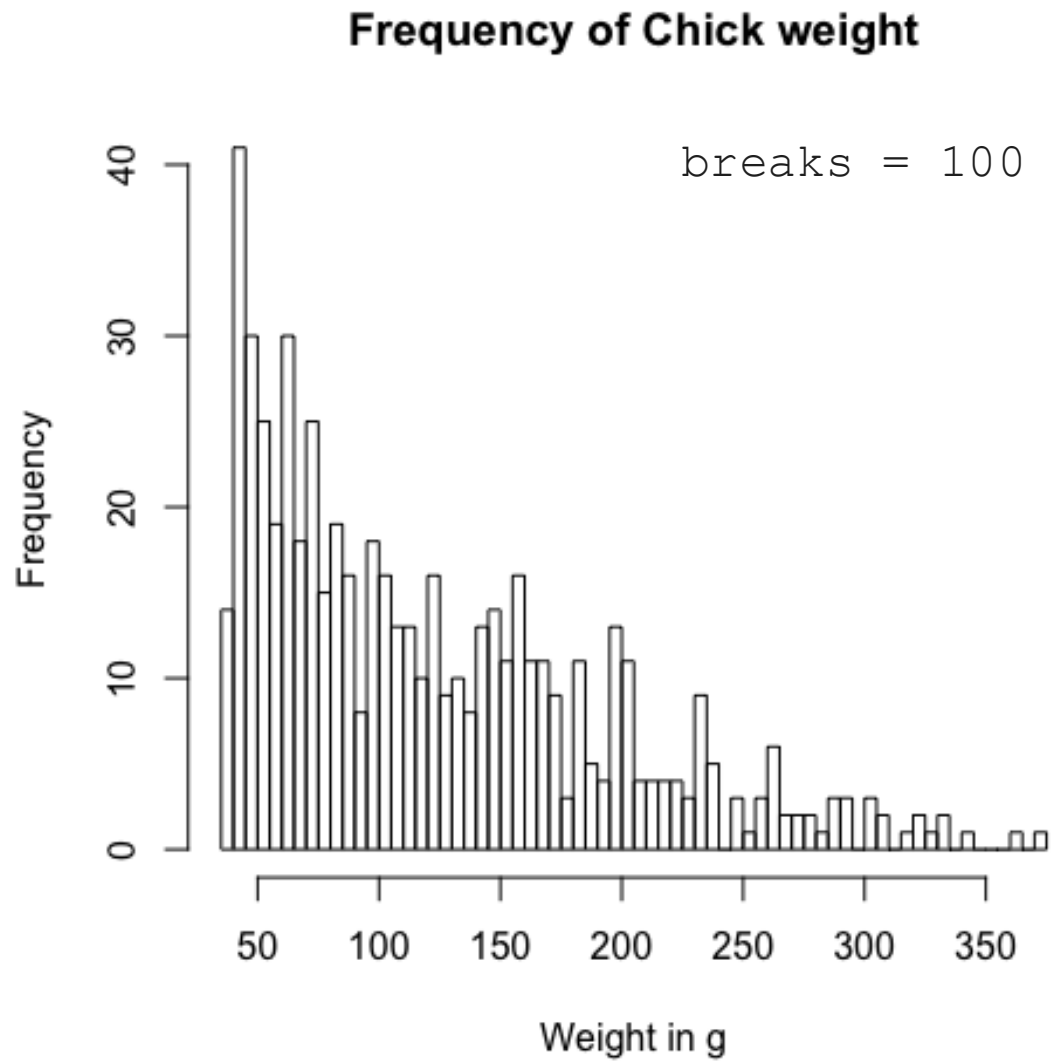
- For example, using the `ChickWeight()` built-in dataset we can plot the frequency of different weights for chicks

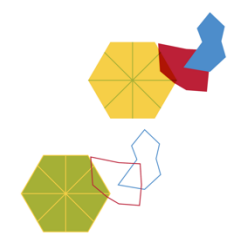
```
> hist(ChickWeight$weight, xlab = "Weight in g", main = "Frequency of Chick weight",
breaks = 20)
>
>
```




Frequency of Chick weight
breaks = 20
Frequency
0
20
40
60
80
100
50
100
150
200
250
300
350
Weight in g




Frequency of Chick weight
breaks = 100
Frequency
0
10
20
30
40
50
100
150
200
250
300
350
Weight in g

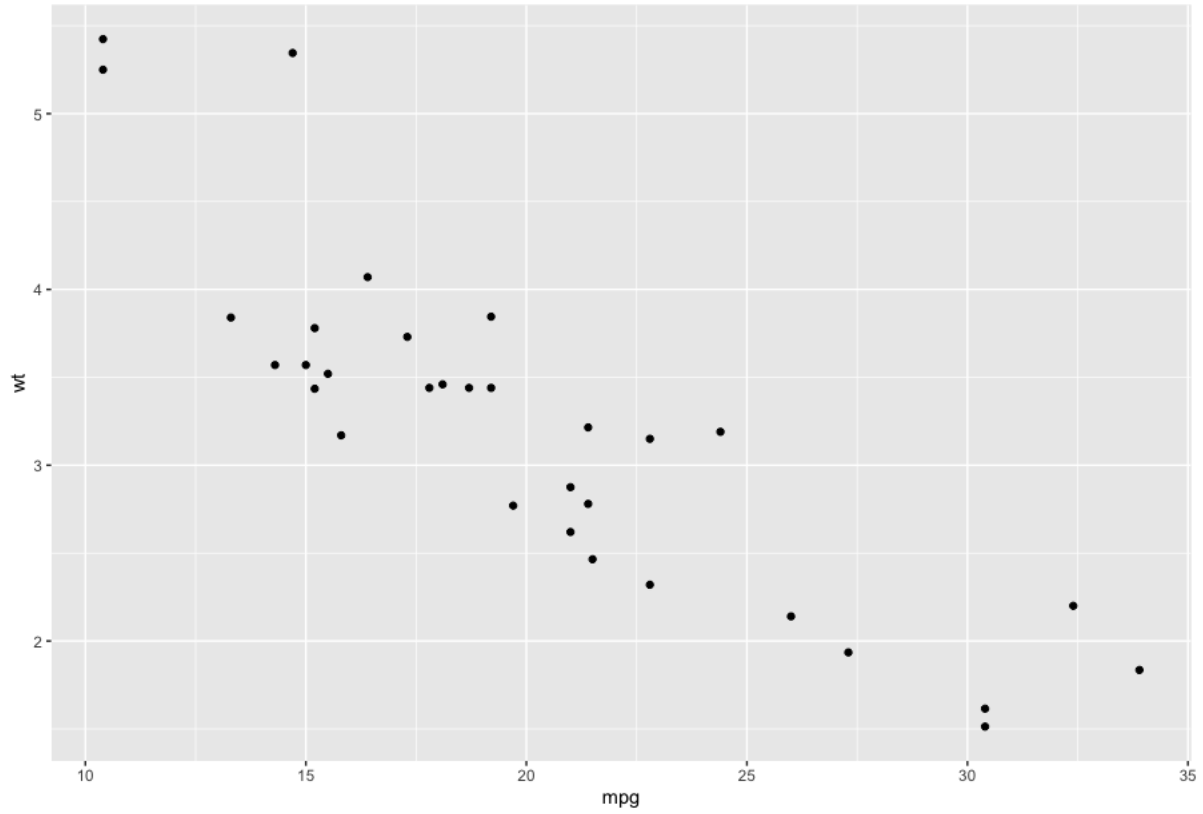


# How to use ggplot2

# Return to basics – How to use ggplot2

- There are 3 components to a ggplot: data, aesthetics, geometry
- Aesthetics and geometries are the two aspects that will give you control

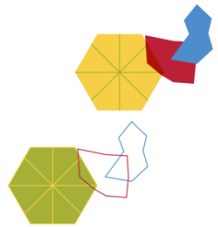
```
1  library(tidyverse)
2  ggplot(mtcars, aes(x=mpg, y=wt)) +
3    geom_point()
```



# Graphics – The `tidyverse` way

Some geometric objects:

- Geometric points for scatterplots: `geom_point()`
- Geometric lines for line graphs: `geom_line()`
- Geometric histograms: `geom_histogram()`



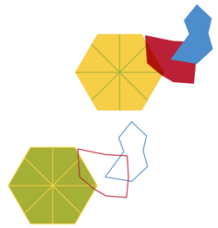
# Graphics – The `tidyverse` way

- Let's start with a scatterplot, using the dataset `diamonds`

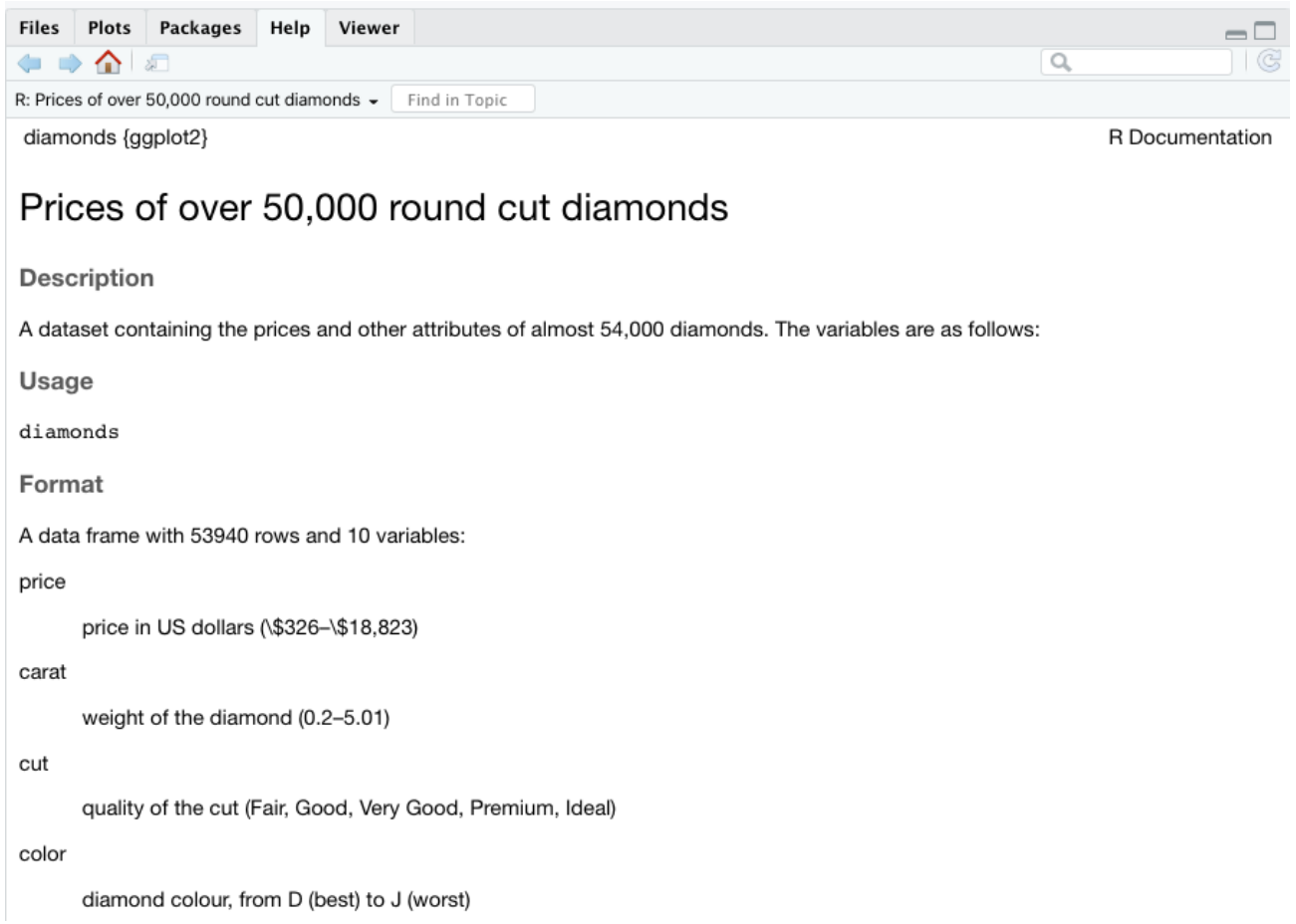
- Use `?diamonds` to find out what's included

Console | Terminal | Jobs

```
> require(tidyverse)
> require(magrittr)
> diamonds
# A tibble: 53,940 x 10
   carat cut       color clarity depth table price     x     y     z
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
 5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
 6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
 7 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
 8 0.26  Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
 9 0.22  Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
10 0.23  Very Good H     VS1      59.4    61   338  4     4.05  2.39
# … with 53,930 more rows
```



# Graphics – The `tidyverse` way



Files | Plots | Packages | Help | Viewer

R: Prices of over 50,000 round cut diamonds — Find in Topic

diamonds {ggplot2} — R Documentation

## Prices of over 50,000 round cut diamonds

### Description

A dataset containing the prices and other attributes of almost 54,000 diamonds. The variables are as follows:

### Usage

```
diamonds
```

### Format

A data frame with 53940 rows and 10 variables:

price

: price in US dollars (\$326–\$18,823)

carat

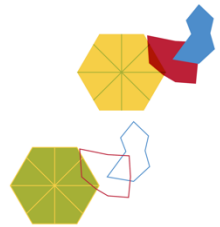
: weight of the diamond (0.2–5.01)

cut

: quality of the cut (Fair, Good, Very Good, Premium, Ideal)

color

: diamond colour, from D (best) to J (worst)

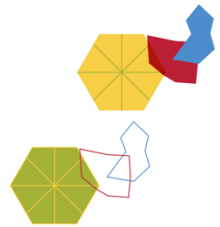


# Graphics – The `tidyverse` way

- Let's start with a scatterplot, using the dataset `diamonds`

- First: `filter` the dataset to only include diamonds of `color` category "`E`".

```
diamonds %>% filter(color=="E")
```

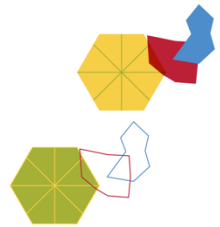


# Graphics – The `tidyverse` way

Then create a `ggplot` object with the aesthetics you want:

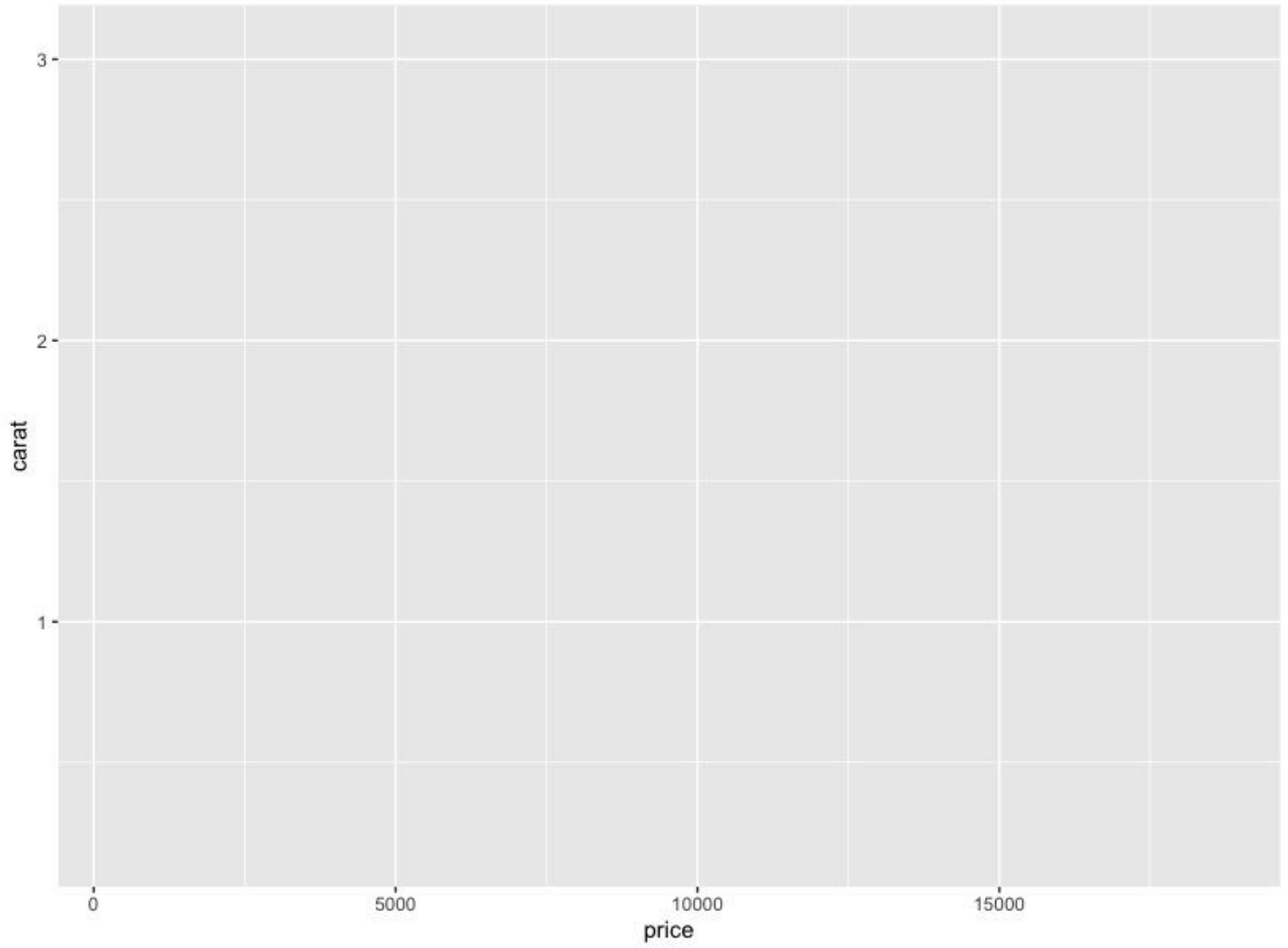
- I would like to create a scatterplot using `price` as the x-axis and `carat` as the y-axis, using different `cuts` as different colours

```
ggplot(diamonds %>% filter(color=="E"),
    aes(x=price,y=carat,colour=cut))
```



# Graphics – The `tidyverse` way

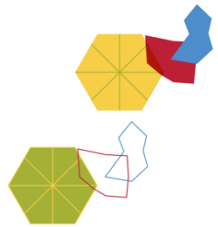
Disappointing…



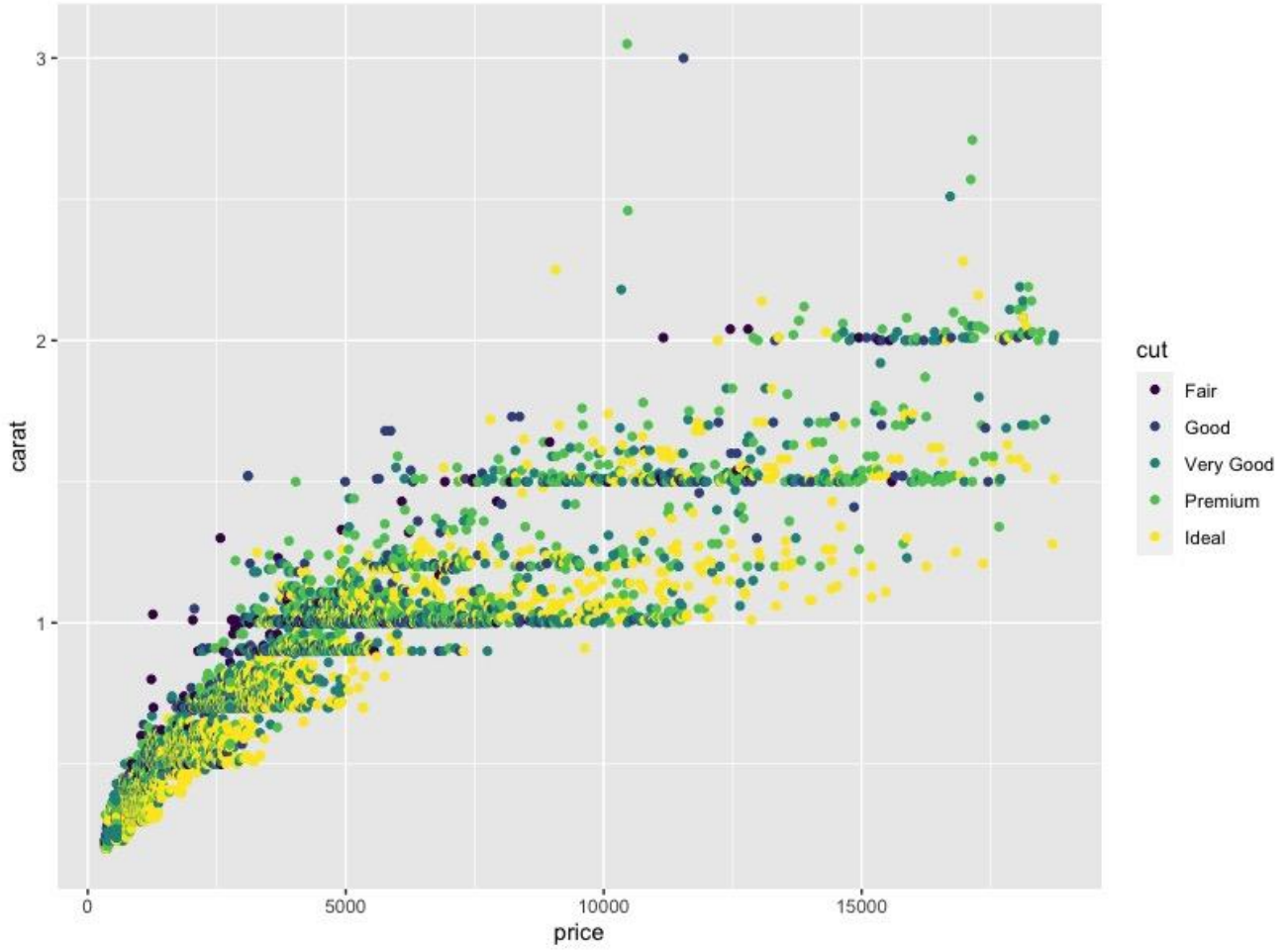
# Graphics – The `tidyverse` way

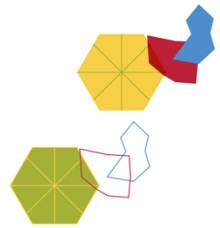
Let's add the geometric object for scatterplots:

```
ggplot(diamonds %>% filter(color=="E"),
                aes(x=price,y=carat,colour=cut)) +
                    geom_point()
```



# Graphics – The `tidyverse` way

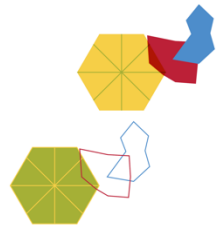


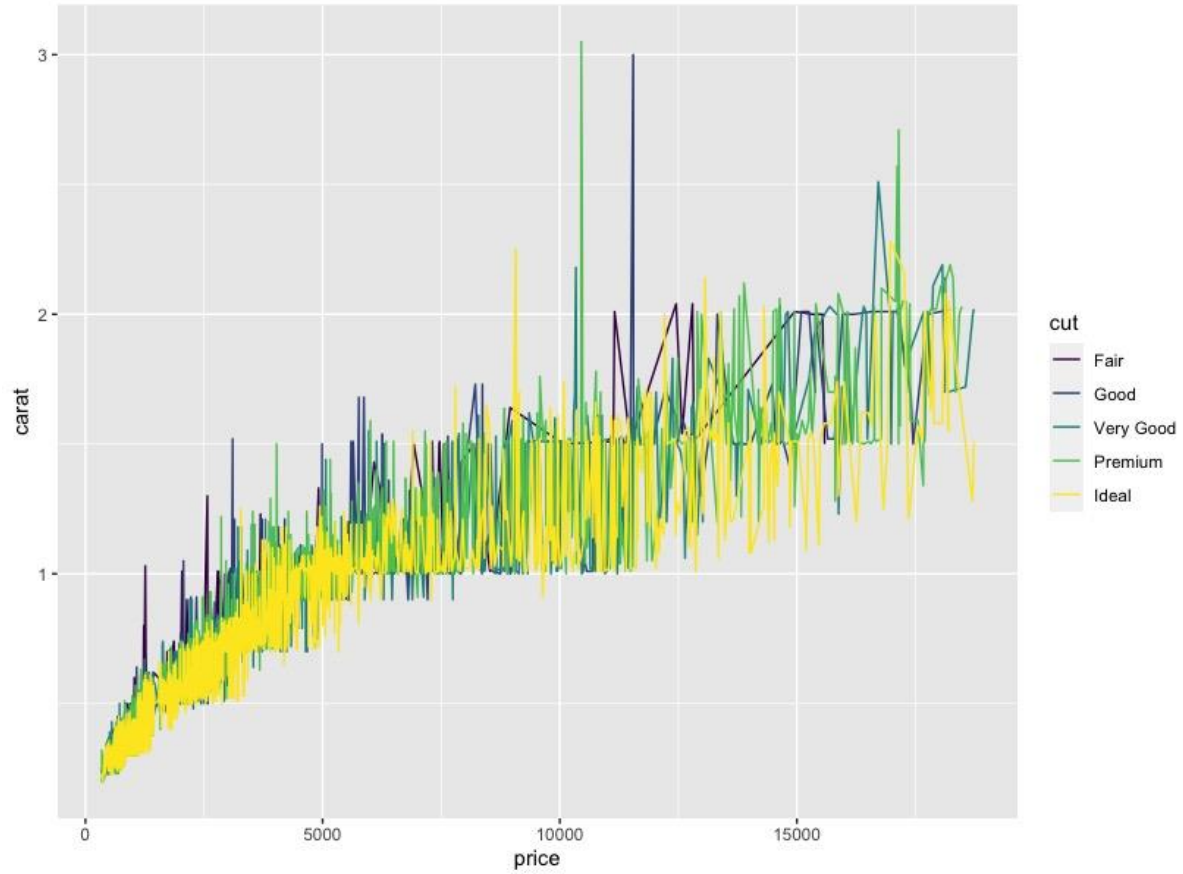
# Graphics – The `tidyverse` way

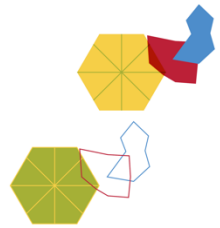
What happens if instead of geom_point() you use geom_line()?

```
ggplot(diamonds %>% filter(color=="E"),
       aes(x=price,y=carat,colour=cut)) +
                    geom_line()
```



# Graphics – The `tidyverse` way

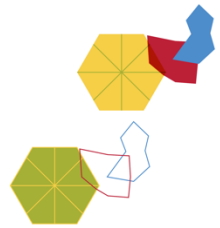


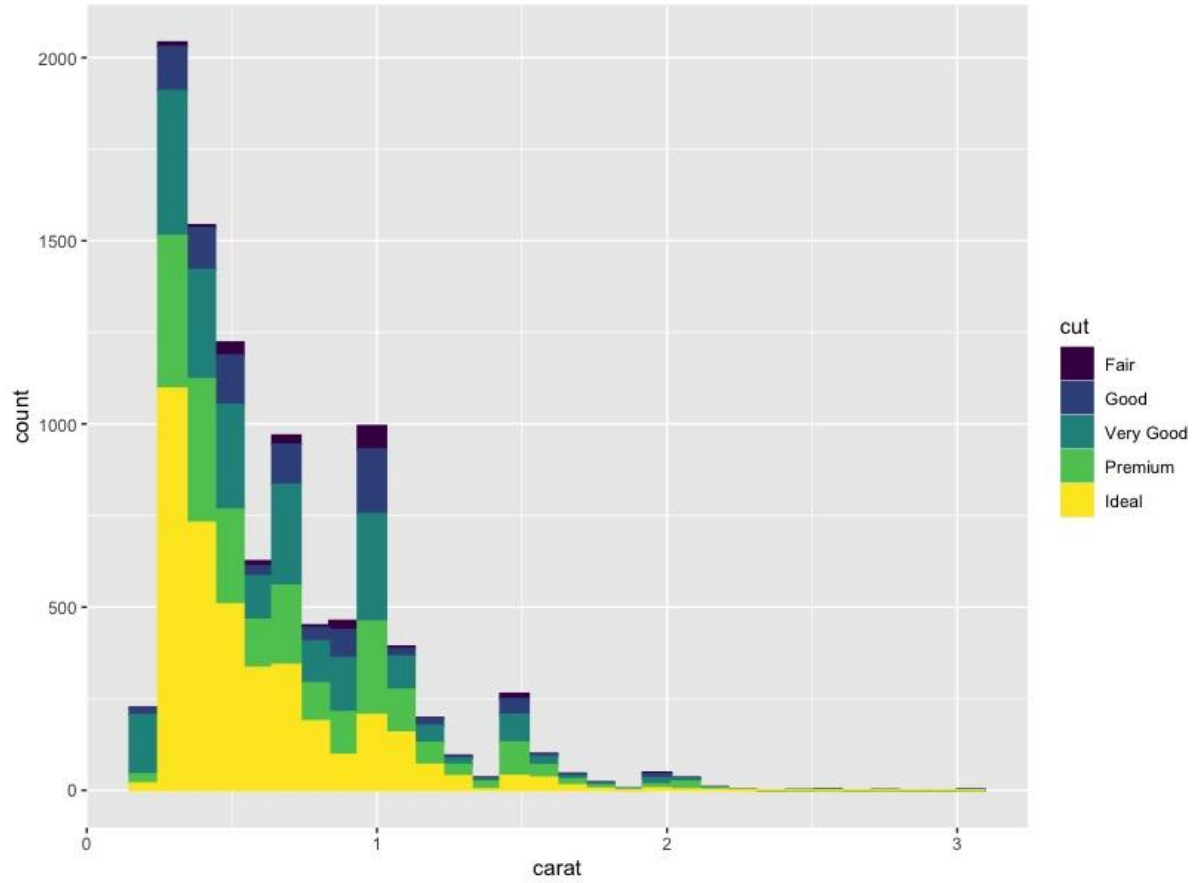
# Graphics – The `tidyverse` way

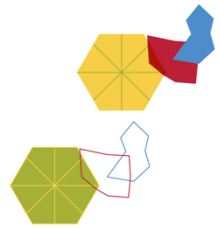
Let us create a histogram for the `carat` size in the reduced `diamonds` dataset using `cut` for colours

```
ggplot(diamonds %>% filter(color=="E"),
  aes(x=carat, colour=cut, fill=cut))+
        geom_histogram()
```



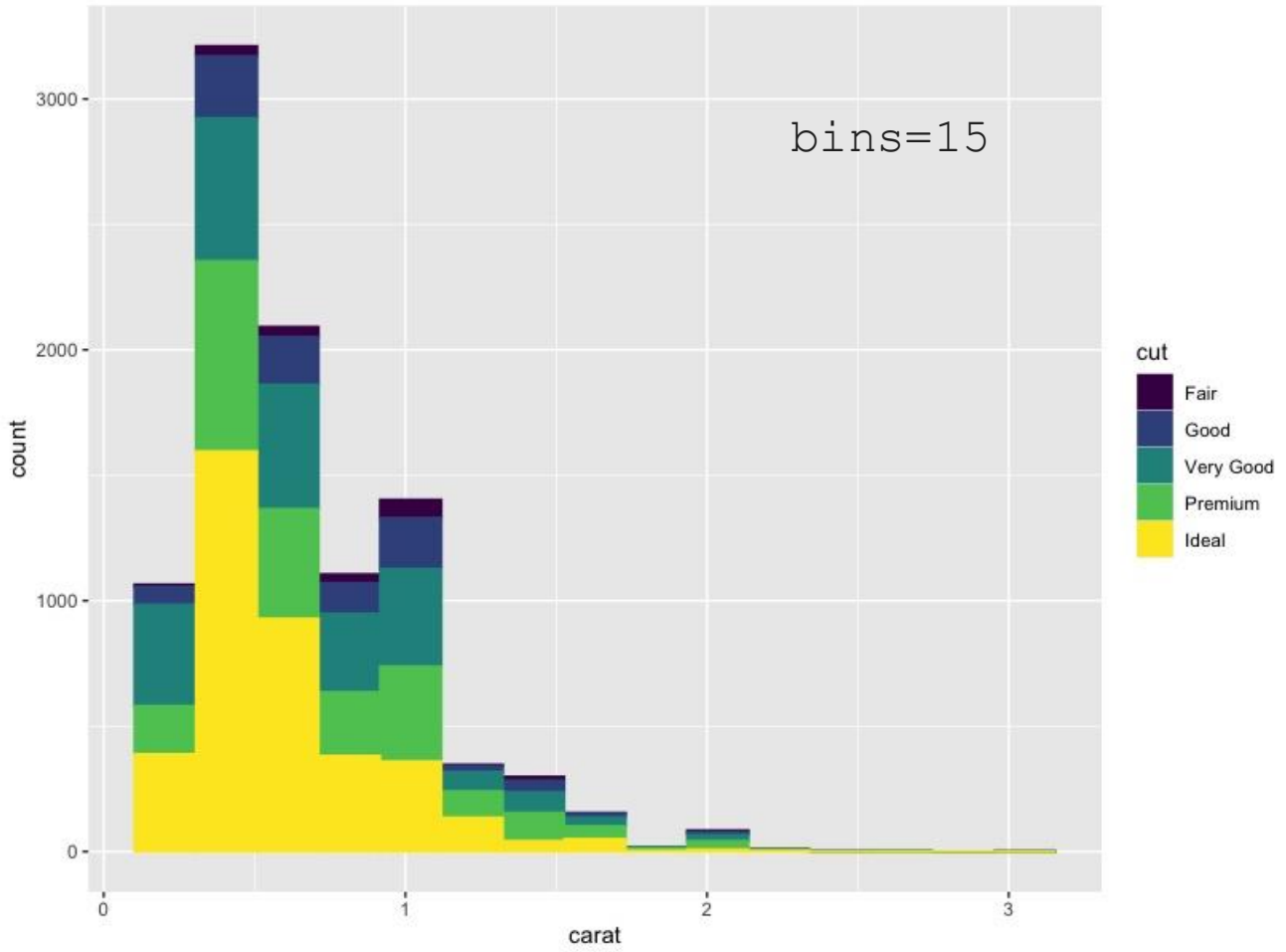
# Graphics – The `tidyverse` way



# Graphics – The `tidyverse` way

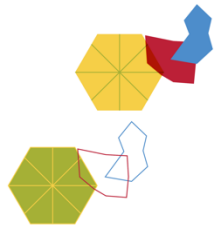
You can change the bin count by adding `bins=` and the number you want in `geom_histogram()`



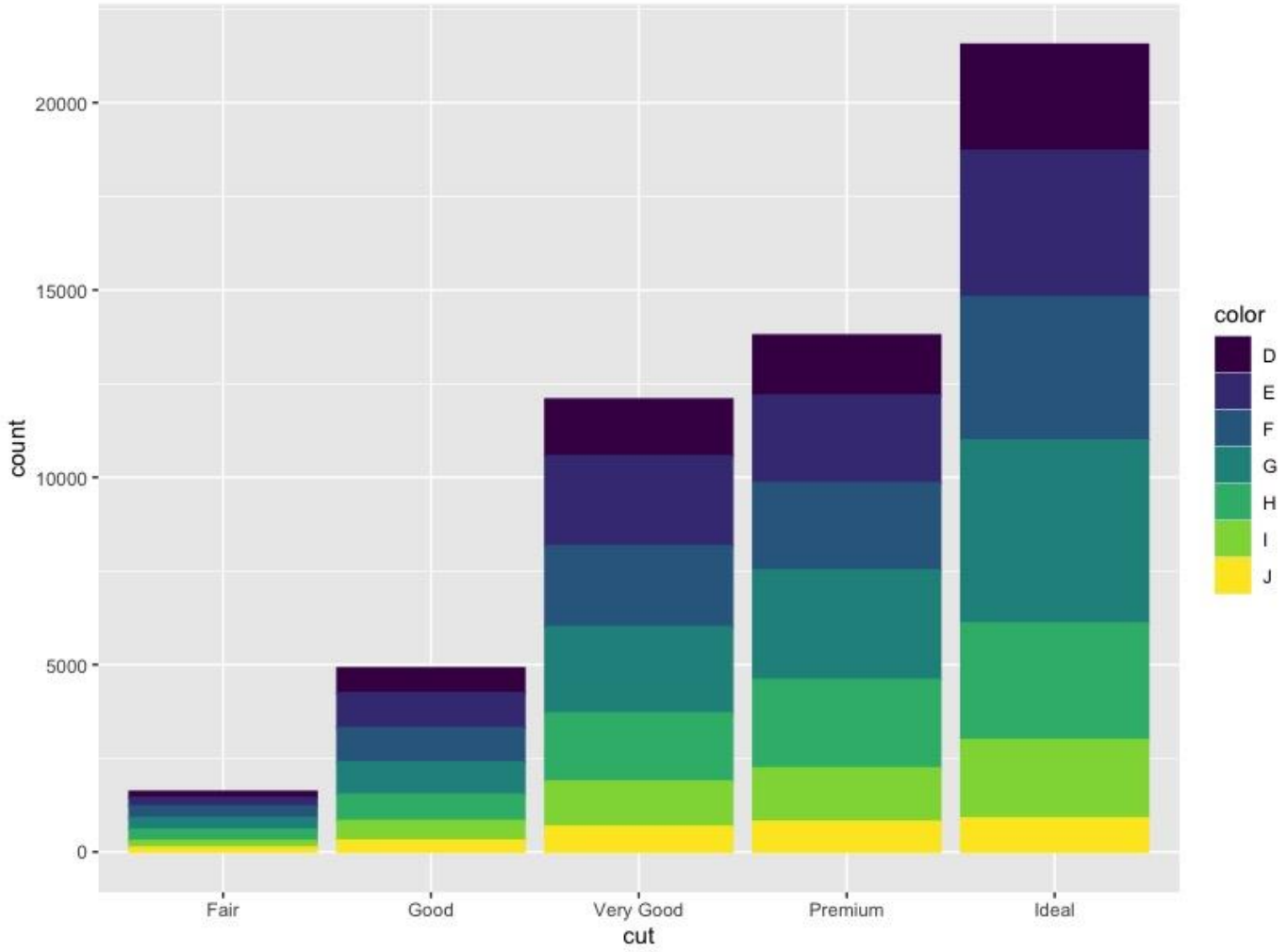
# Graphics – The `tidyverse` way

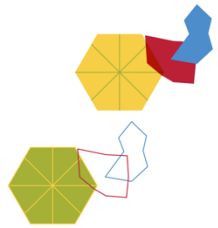
Let us create a barplot for the `cuts` in the original `diamonds` dataset using the `color` variable for colours

```
ggplot(diamonds, aes(x=cut, colour=color,
        fill=color))+  geom_bar()
```



# Graphics – The `tidyverse` way

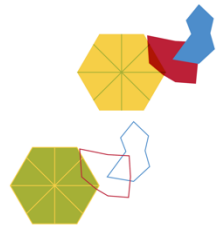


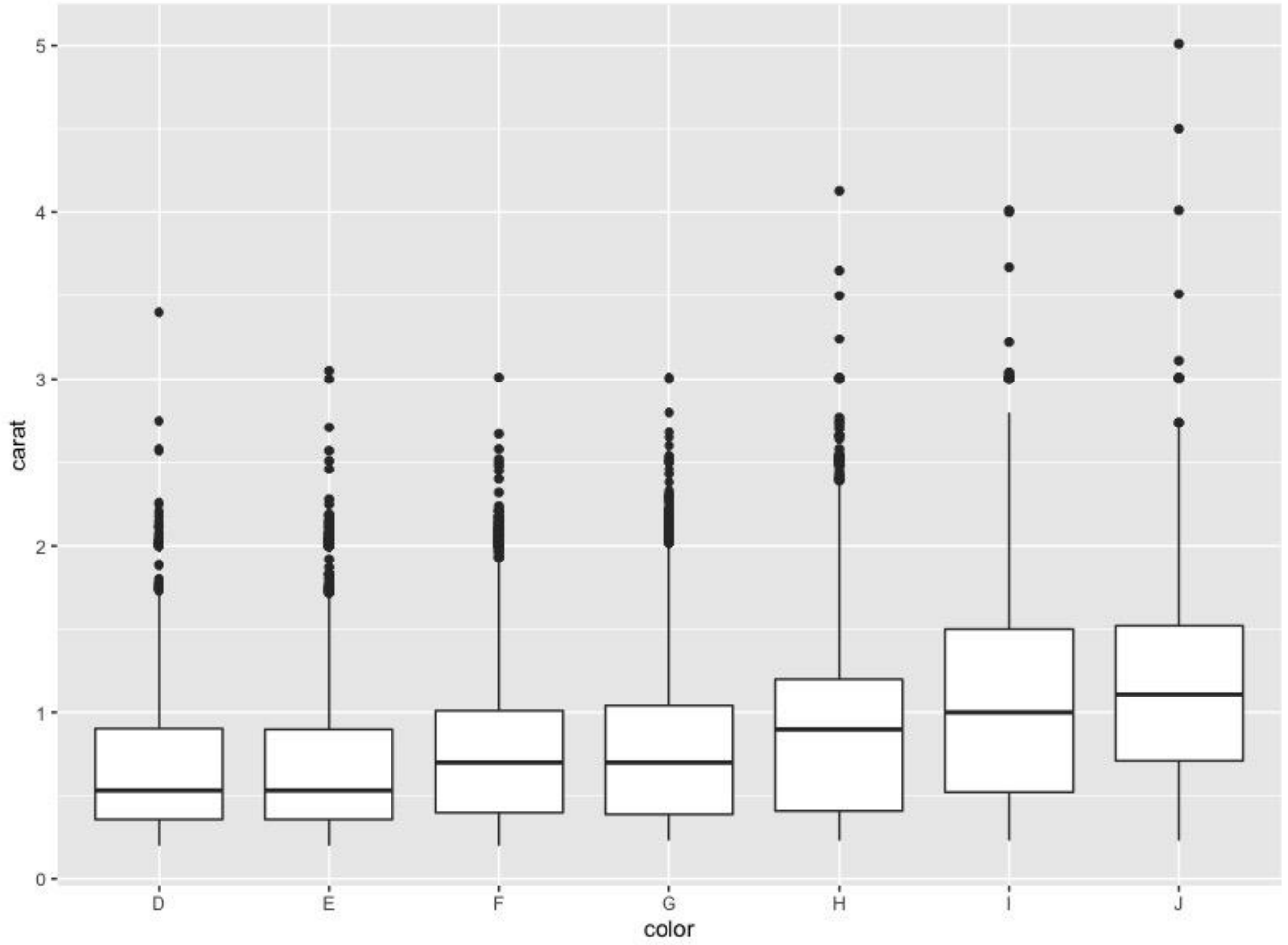
# Graphics – The `tidyverse` way

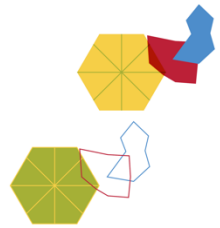
**Let us create a boxplot for the `carats` by `color` in the original `diamonds` dataset**

```
ggplot(diamonds,aes(x=color,y=carat))+
  geom_boxplot()
```



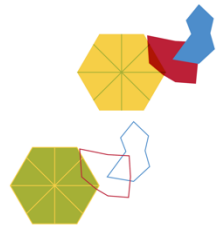
# Graphics – The `tidyverse` way



# Aesthetics

- Colour
  - Colour of object
  - Fill of object
- Lines
  - Type of line
  - Thickness
  - Joins between lines
  - Line ends
- Shape
  - Shape of the points
- Text
  - Annotations
  - Font face
  - Font type
  - Font size
  - Justification



# Colours

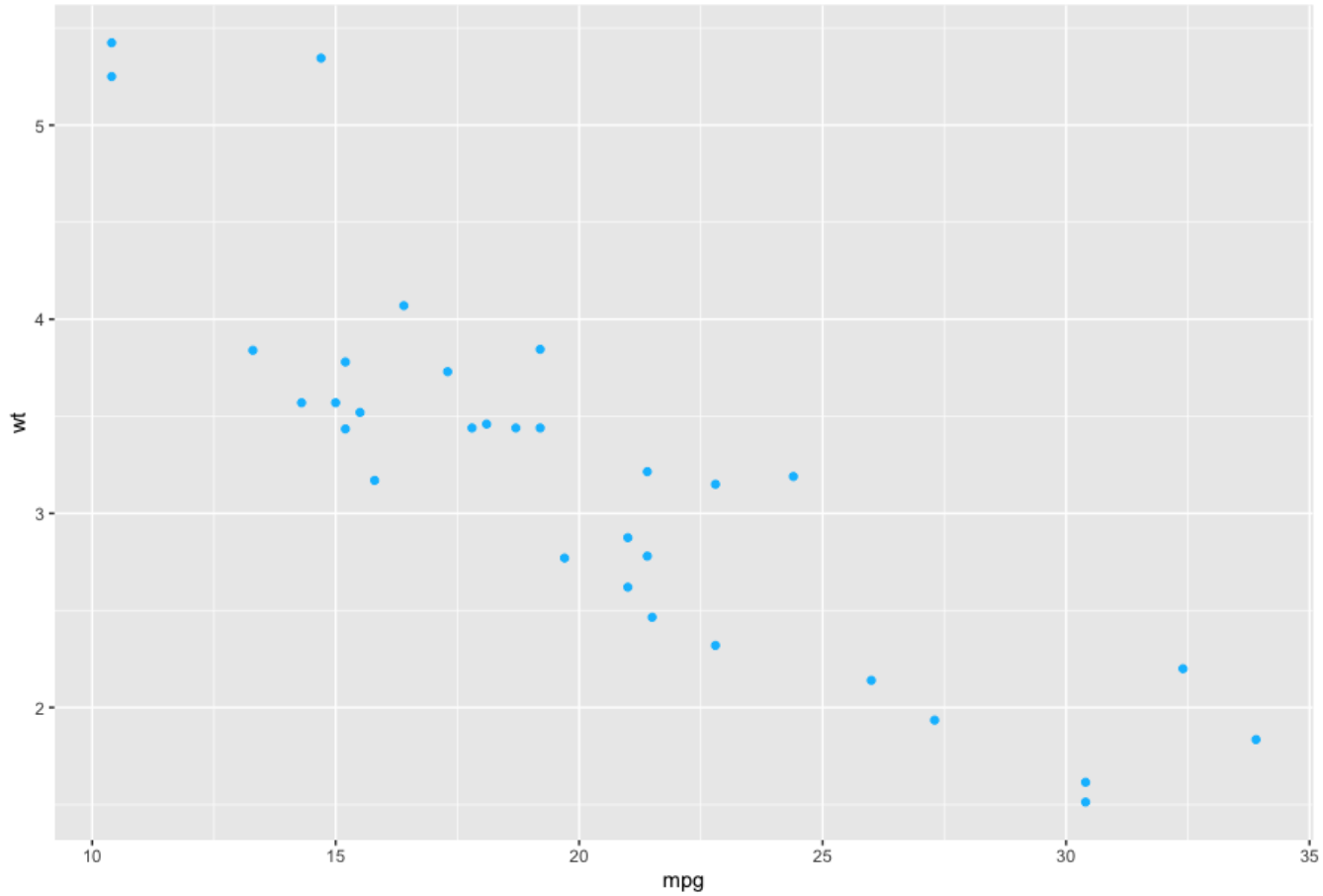
| color | name |
|---|---|
| | gray48 |
| | gray49 |
| | gray50 |
| | gray51 |
| | gray52 |
| | gray53 |
| | gray54 |
| | gray55 |
| | gray56 |
| | gray57 |
| | gray58 |
| | gray59 |
| | gray60 |
| | gray61 |
| | gray62 |
| | gray63 |
| | gray64 |
| | gray65 |

- Even in base R there are plenty of colours to choose from
- Colours can be specified either through their name (e.g. “deepskyblue1”) or using a hex code (e.g. “#33B2FF”)

| color | name | color | name |
|---|---|---|---|
| | darkgreen | | deepskyblue |
| | darkgrey | | deepskyblue1 |
| | darkkhaki | | deepskyblue2 |
| | darkmagenta | | deepskyblue3 |

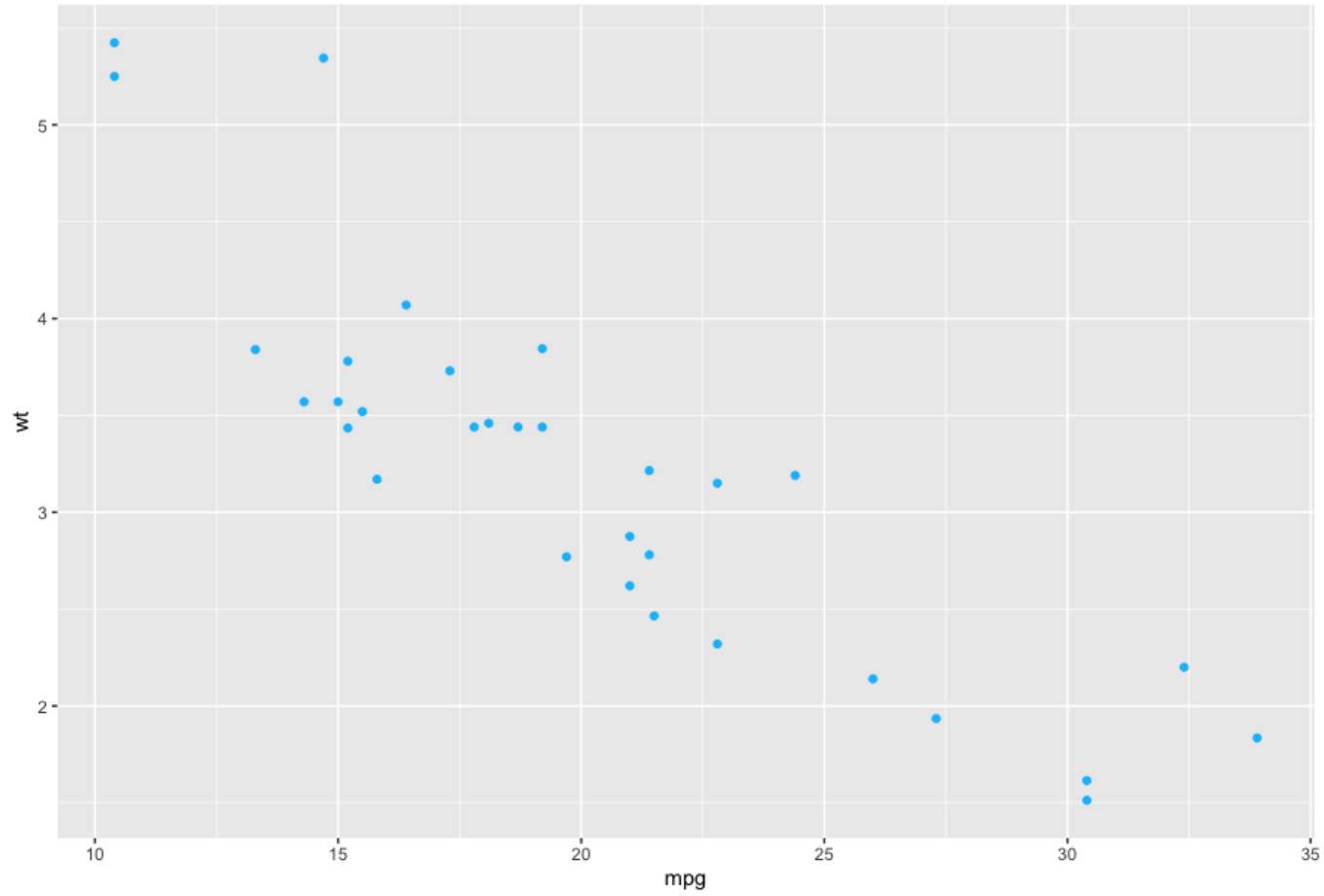
# Colours

```
ggplot(mtcars, aes(x=mpg, y=wt)) +
  geom_point(colour="deepskyblue1")
```



# Colours

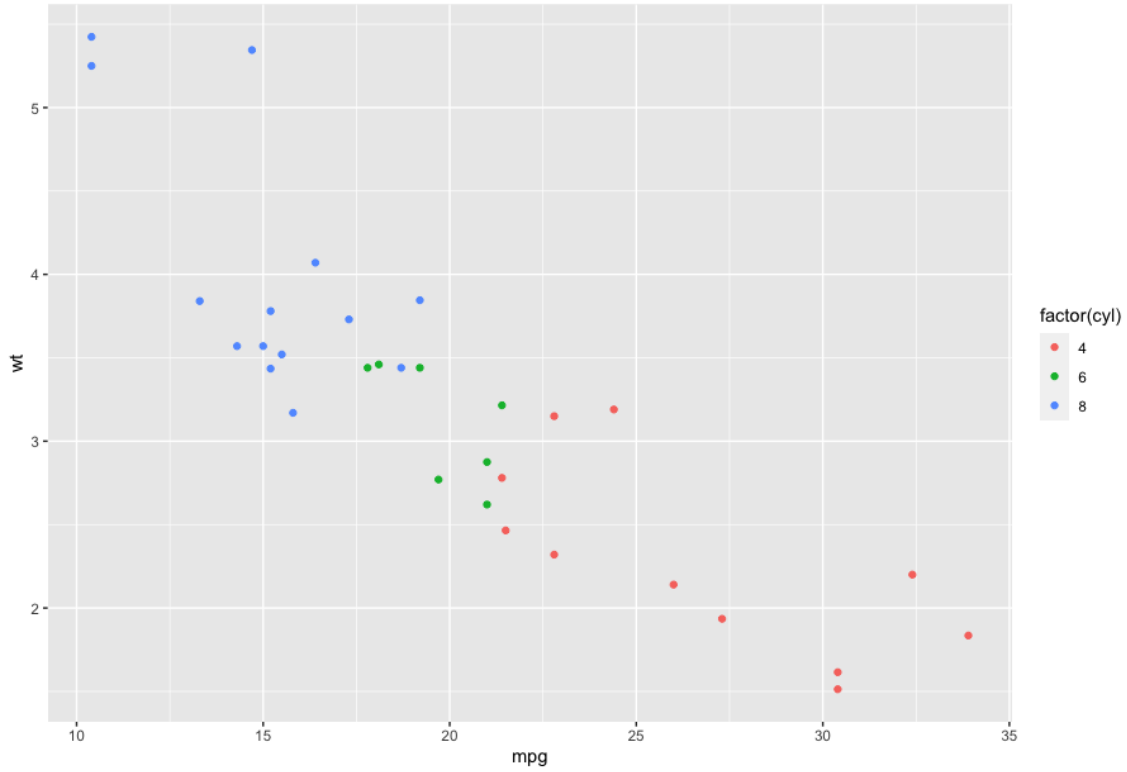
```
ggplot(mtcars, aes(x=mpg, y=wt)) +
  geom_point(colour="#33B2FF")
```



# Colours

- When trying to colour you need to specify which variable you are colouring by or if you are using a single colour for everything

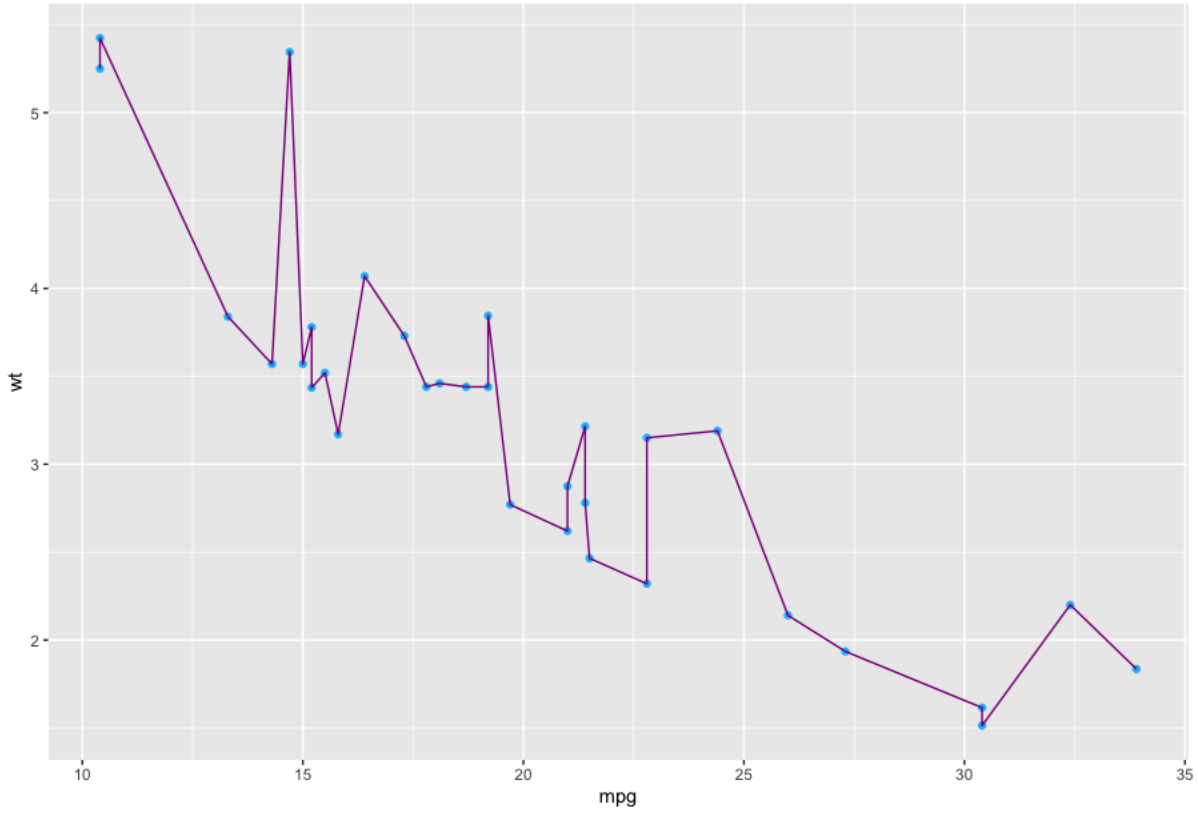
```
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = factor(cyl)))
```



# Colours

- You can also specify different colours for different geometric objects

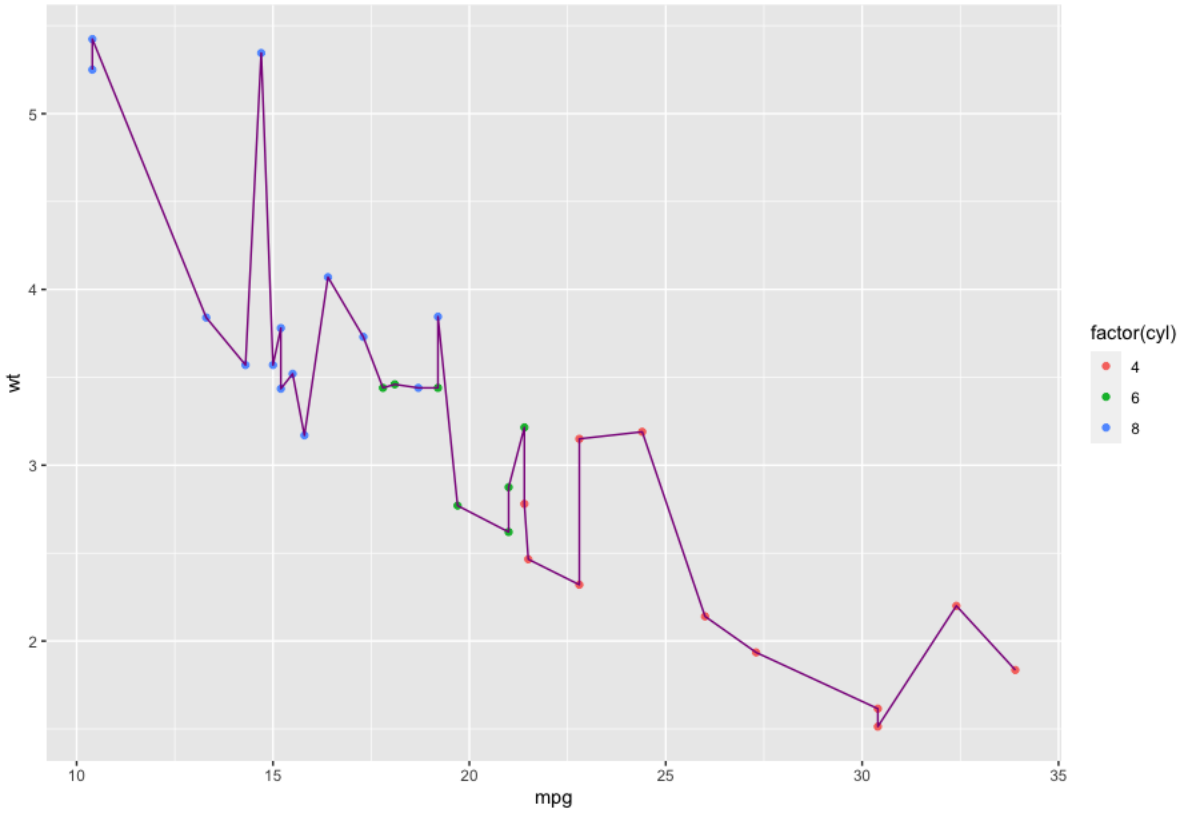
```
ggplot(mtcars, aes(x=mpg, y=wt)) +
  geom_point(colour="deepskyblue1")+
  geom_line(colour="darkmagenta")
```



# Colours

- You can also specify different colours for different geometric objects

```
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = factor(cyl)))+
  geom_line(colour="darkmagenta")
```



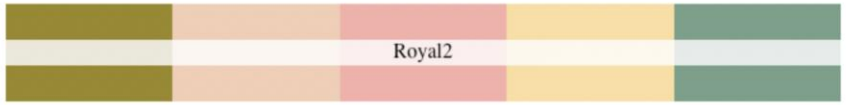
# Colours

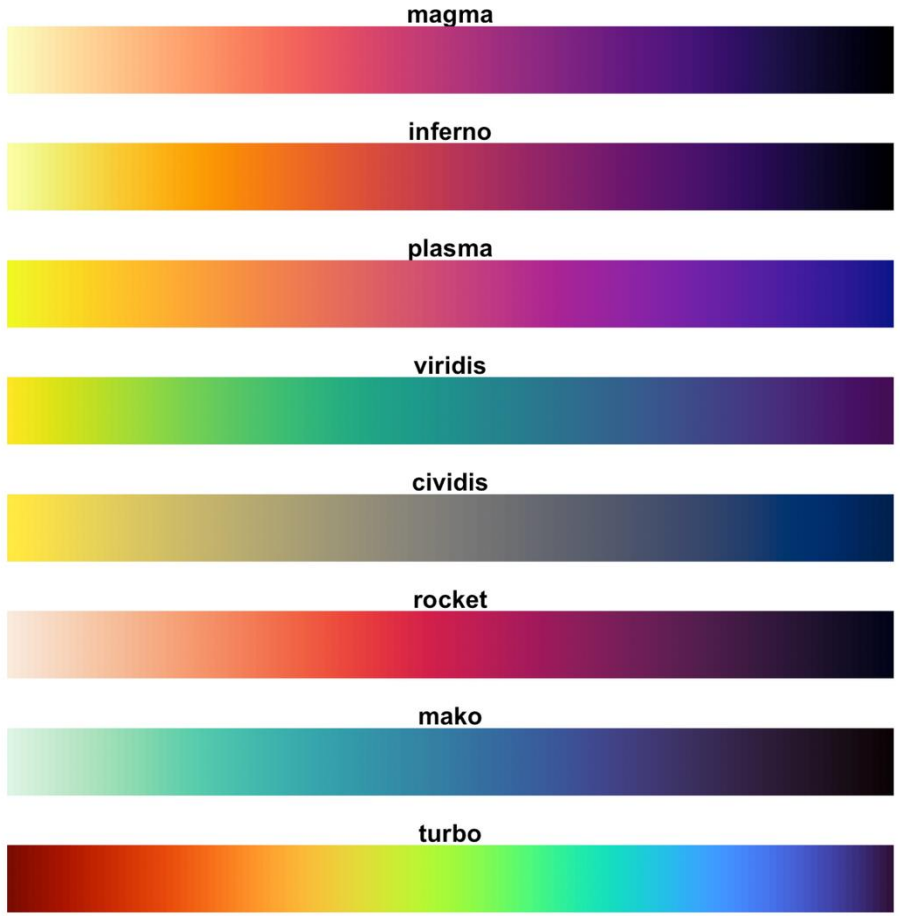
- You can also use a large variety of available colour palettes

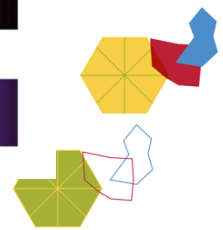
The Royal Tenenbaums (2001)

```
wes_palette("Royal1")
```


```
wes_palette("Royal2")
```

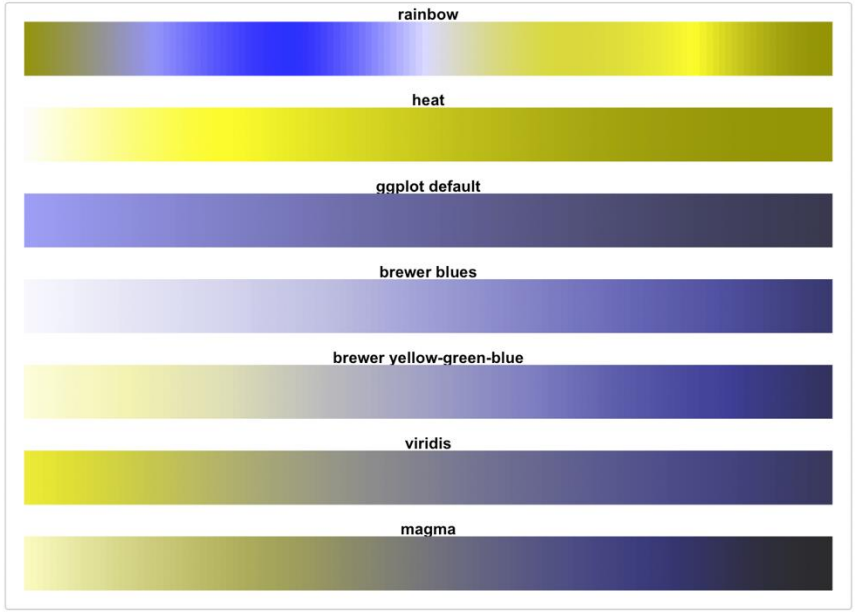


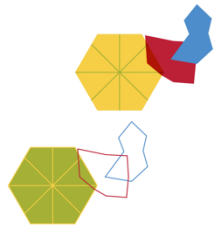



# Colours

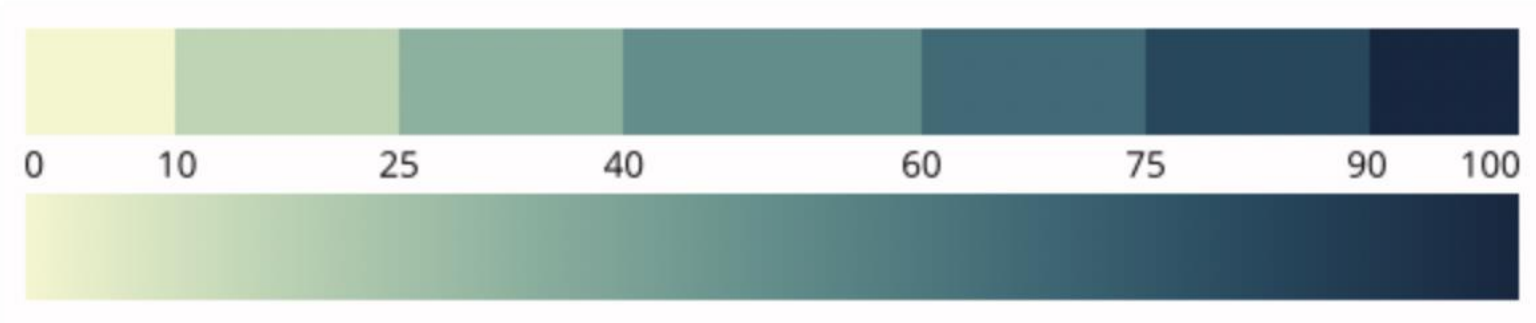
- Viridis and magma colour palettes are very commonly used
- They perform well against other palettes for colour blindness

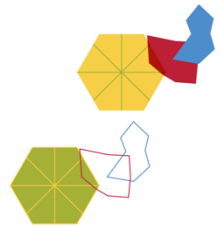



# Colours

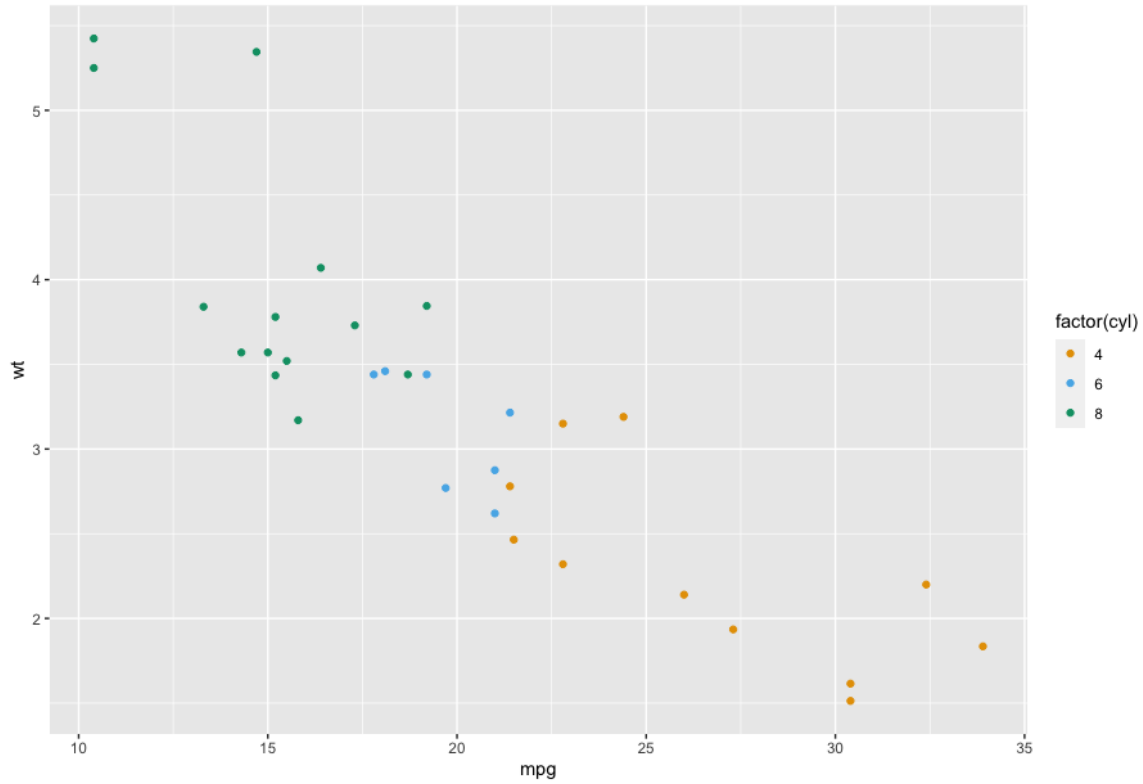
- Using palettes is done through the scales helper functions
- We will focus on two main flavours of the scales functions “discrete” and “continuous”
- Discrete is when we associate a single colour for a range of values and continuous is when we create a gradient of colour from one value to the next

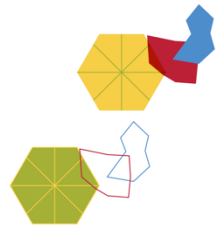


# Colours

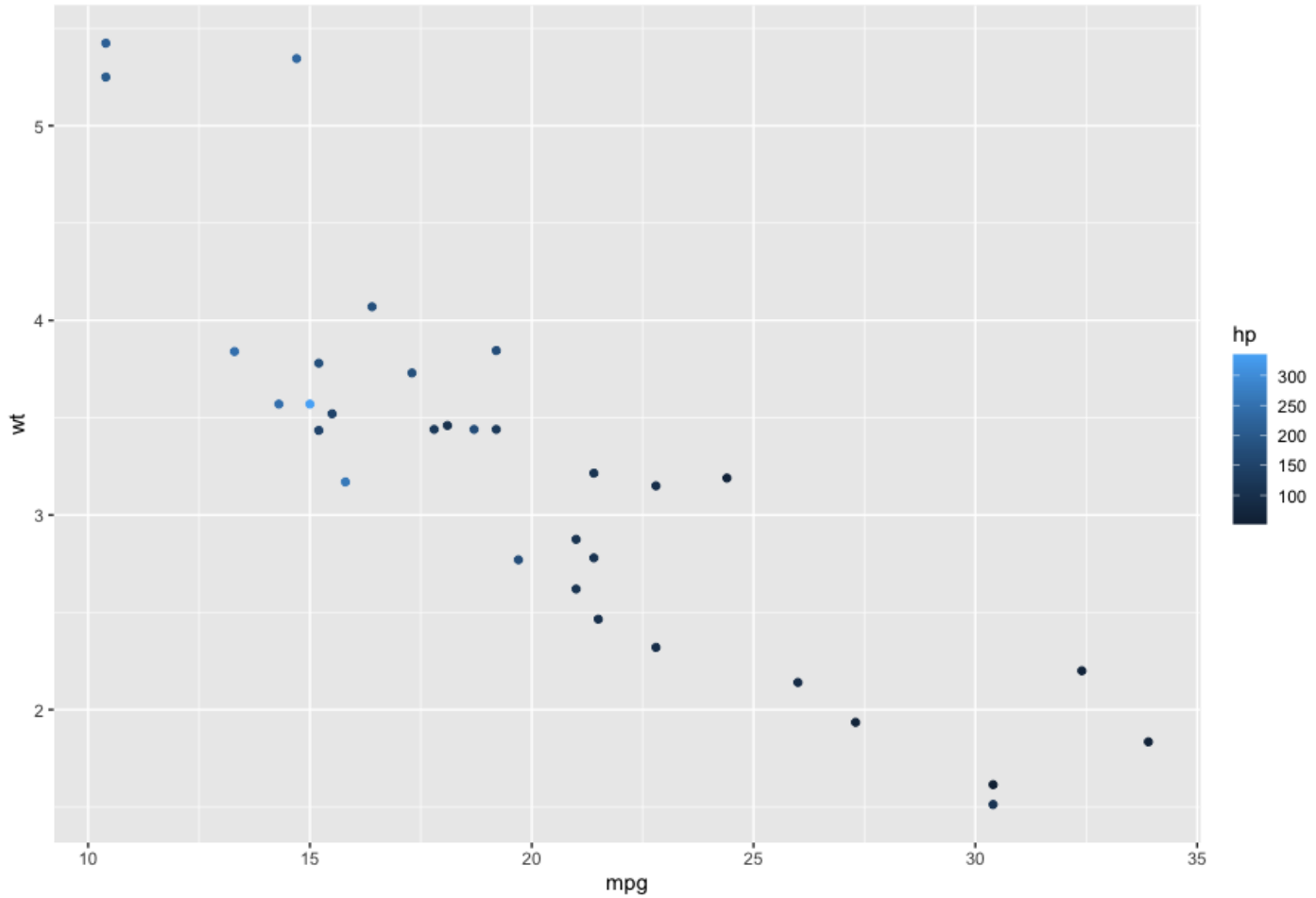
```
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = factor(cyl)))+
  scale_colour_discrete(type=c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7"))
```



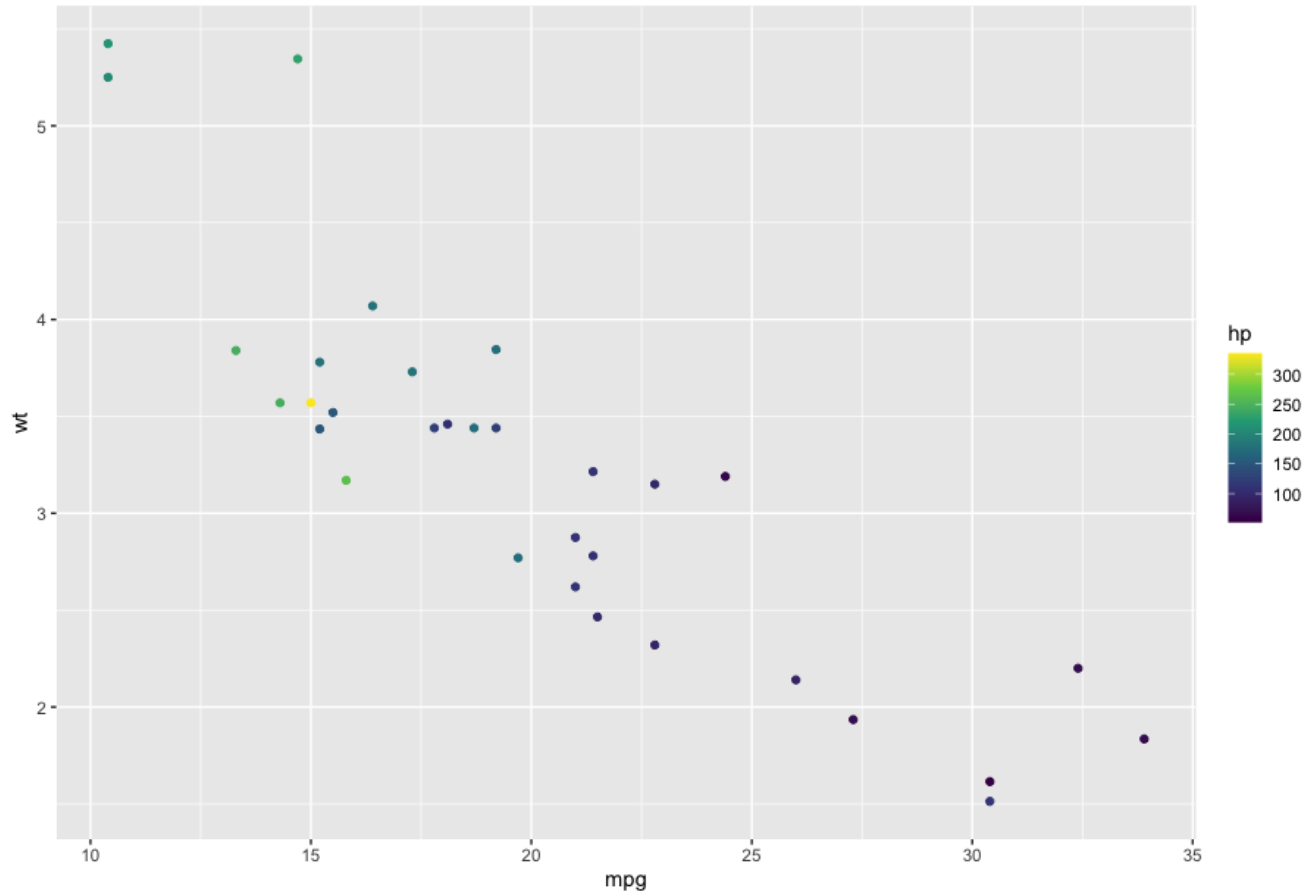
# Colours

```
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = hp))+
  scale_colour_continuous()
```



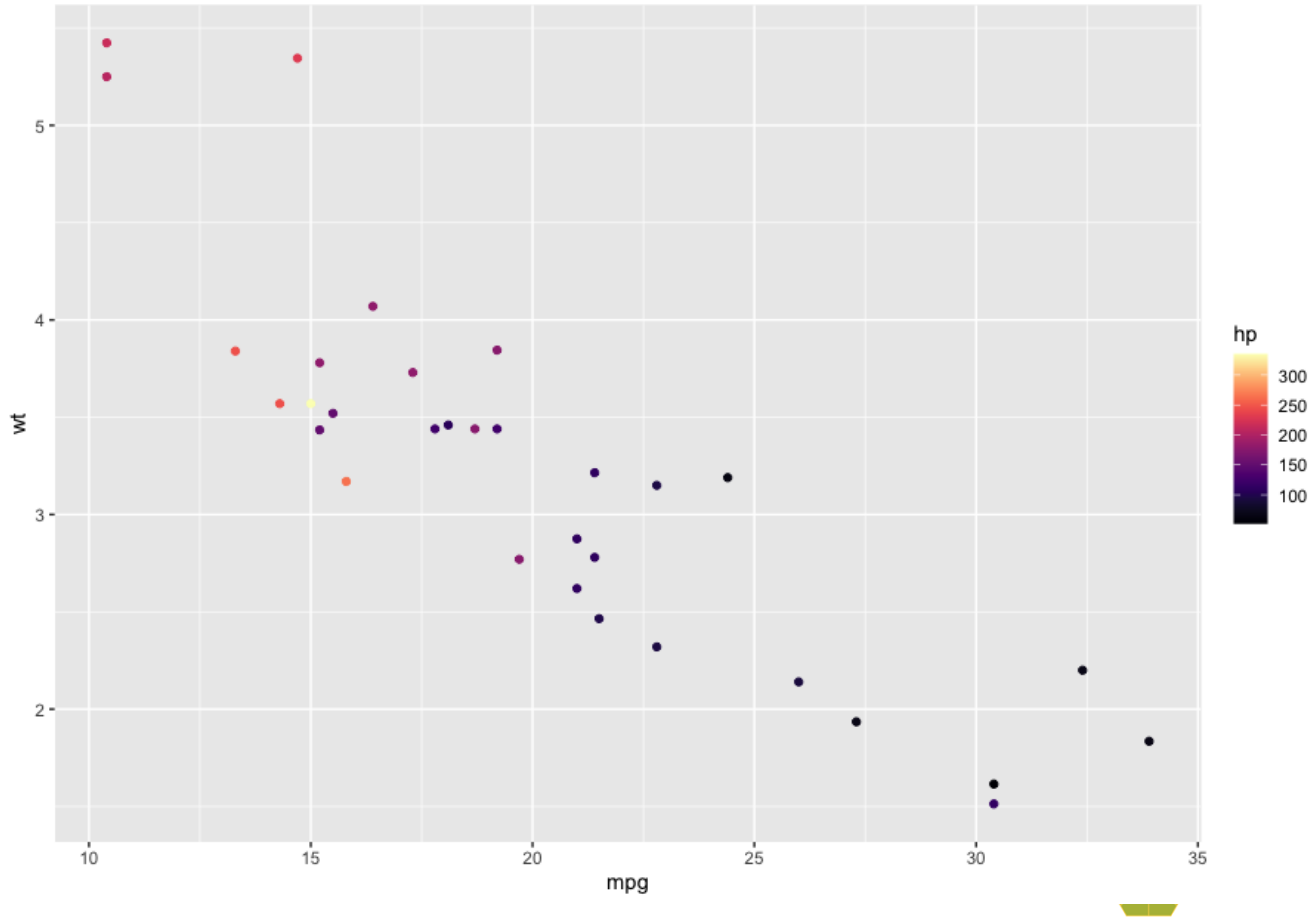
# Colours

```
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = hp))+
  scale_colour_continuous(type="viridis")
```



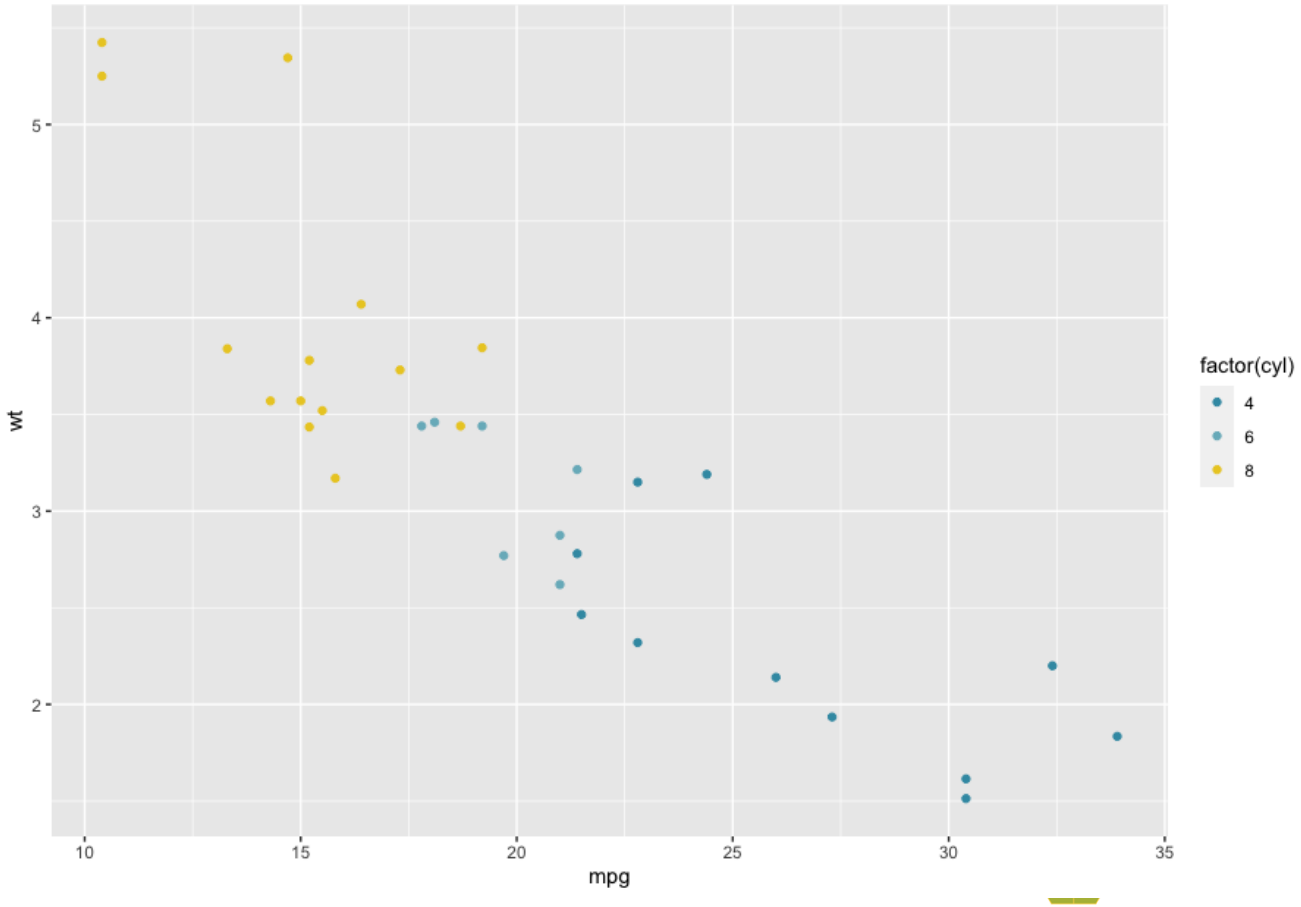
# Colours

```
library(viridis)
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour = hp))+
  scale_colour_viridis(discrete =FALSE, option="magma")
```



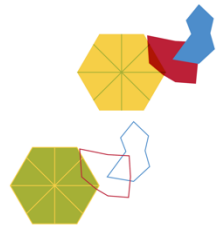
# Colours

```
library(wesanderson)
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(aes(colour=factor(cyl)))+
  scale_colour_manual(values = wes_palette("Zissou1"))
```



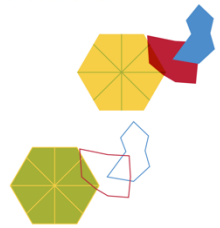
# Lines

- There is a variety of line-types that you can use to specify the exact look of your lines
- You can specify them either by name (e.g. linetype="solid") or by an integer (e.g. linetype=1)

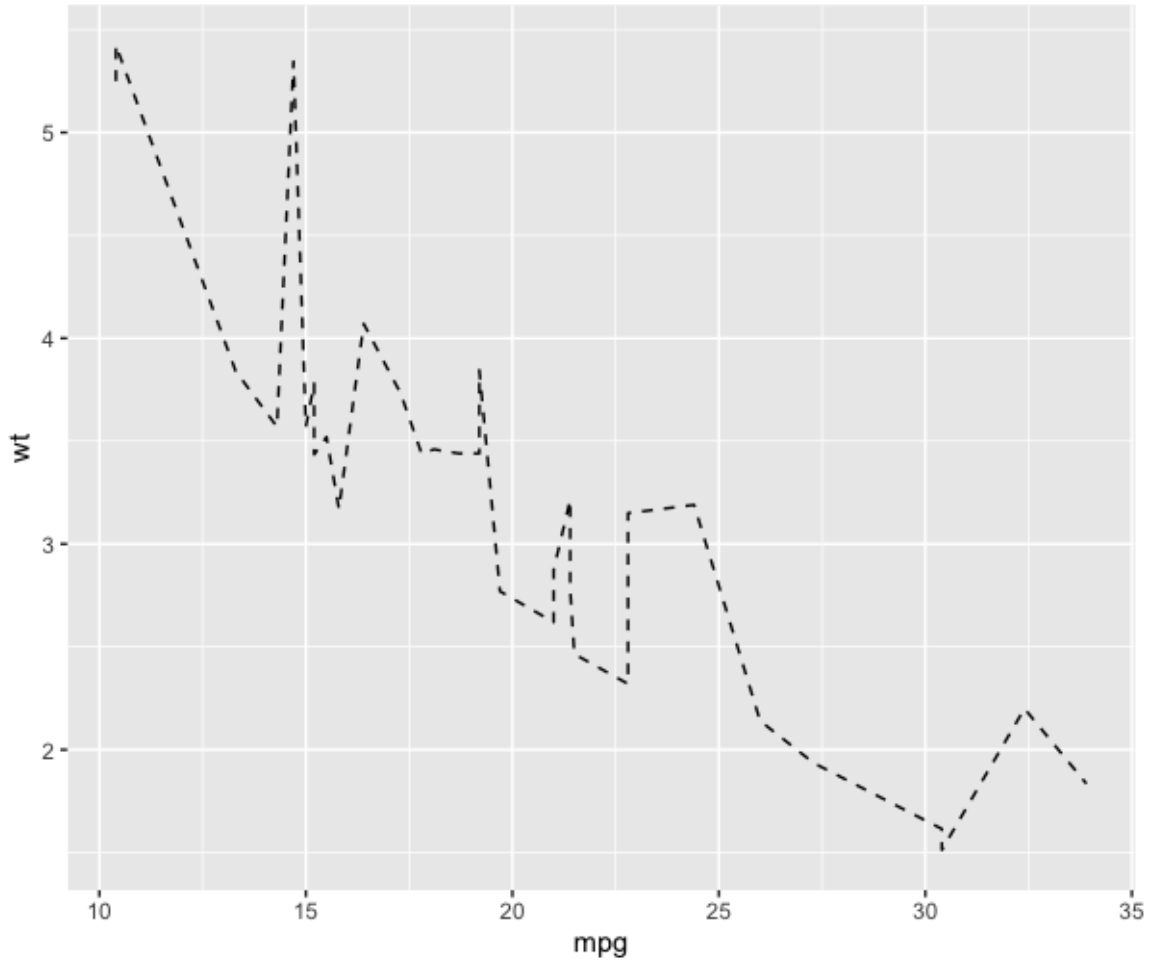
# Lines

- Integers:
- 0 – blank
- 1 – solid
- 2 – dashed
- 3 – dotted
- 4 – dotdash
- 5 – longdash
- 6 – twodash

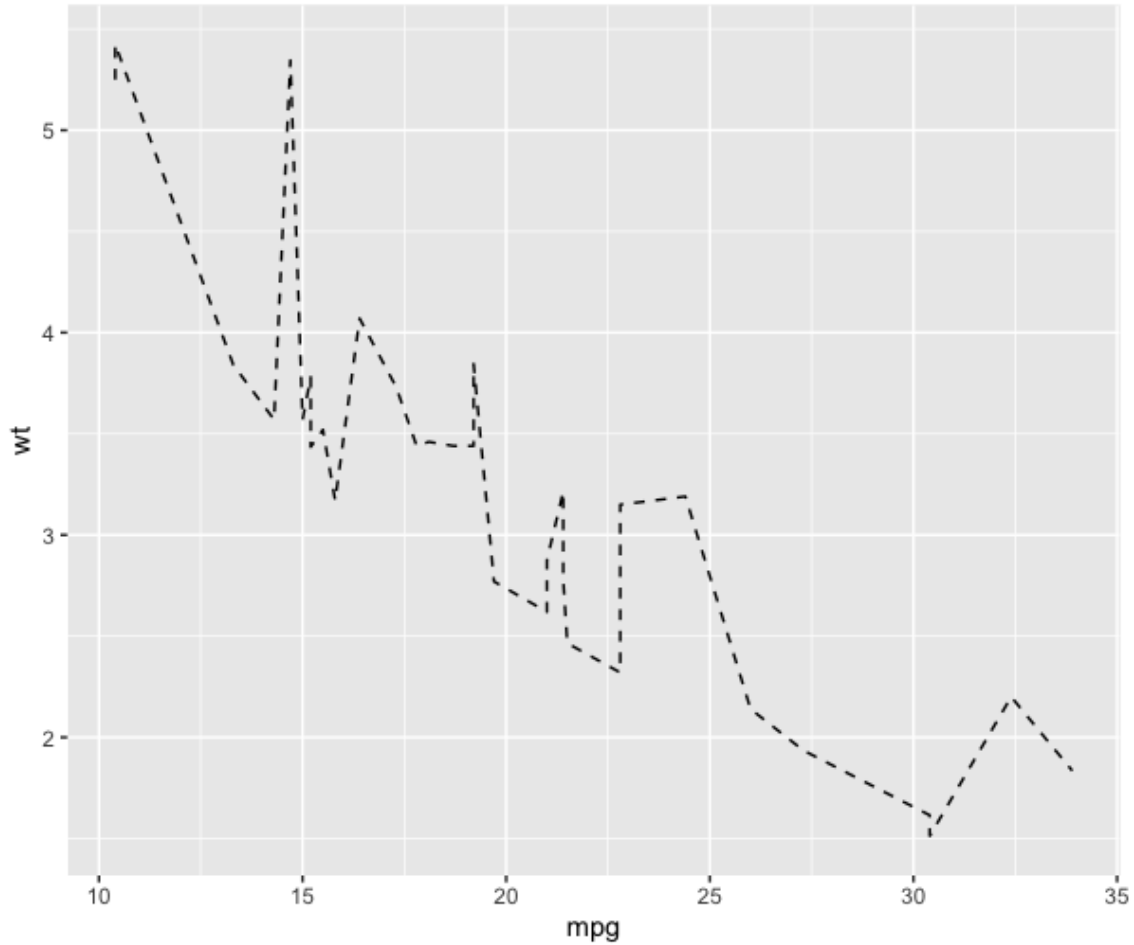
# Lines

```
ggplot(mtcars, aes(mpg, wt)) +
  geom_line(linetype="dashed")
```



# Lines

```
ggplot(mtcars, aes(mpg, wt)) +
  geom_line(linetype=2)
```

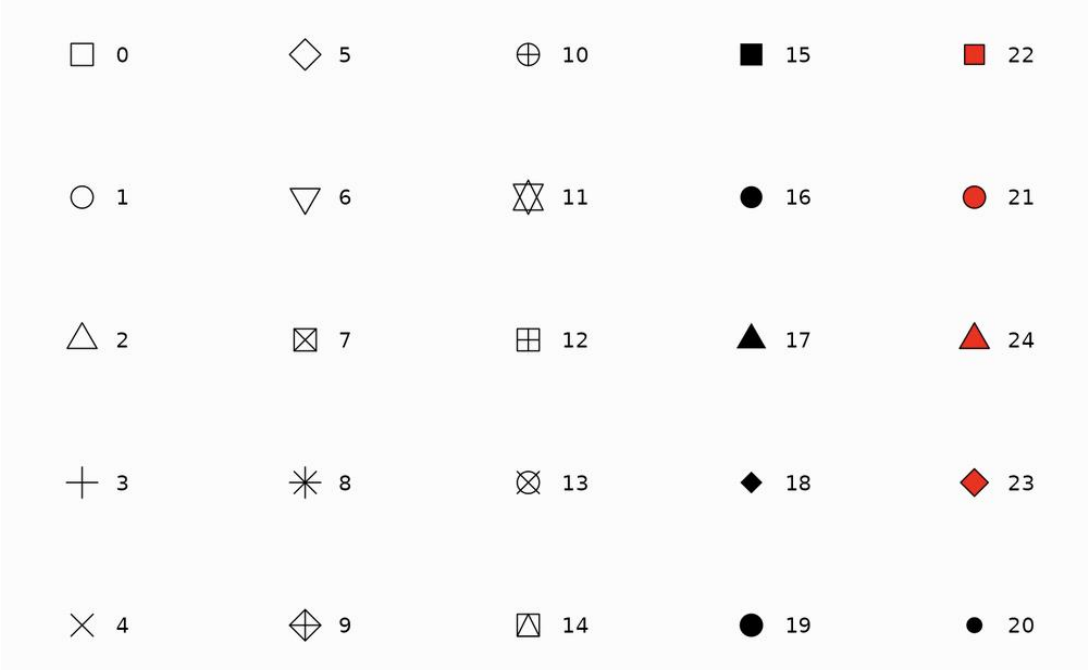


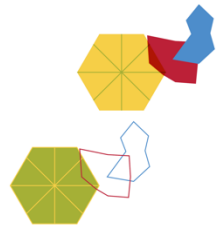
# Shape

- Similar to lines, you can specify point shapes either through name or integer

- E.g. shape="triangle" or

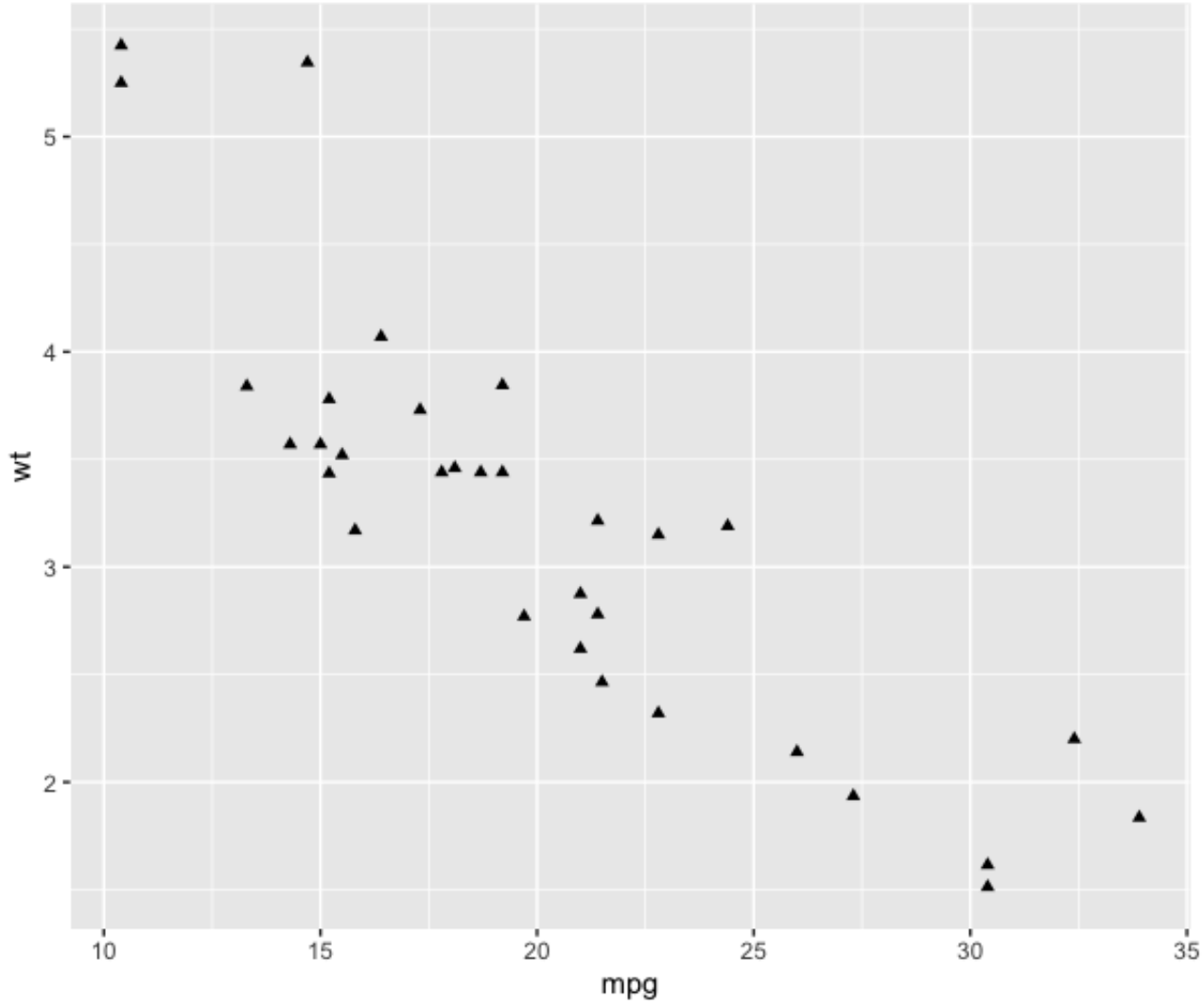
  shape= 17



# Shape

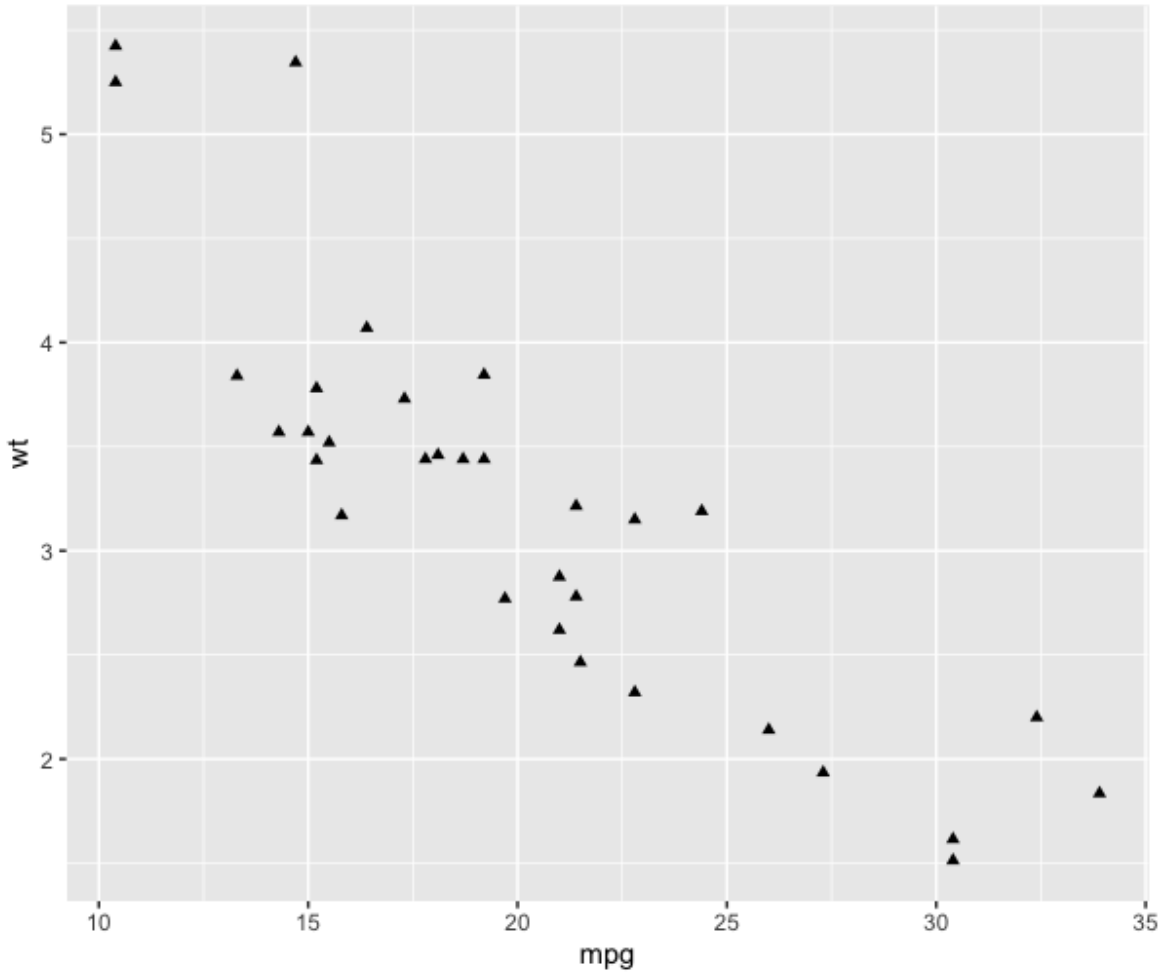
```
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(shape="triangle")
```



# Themes

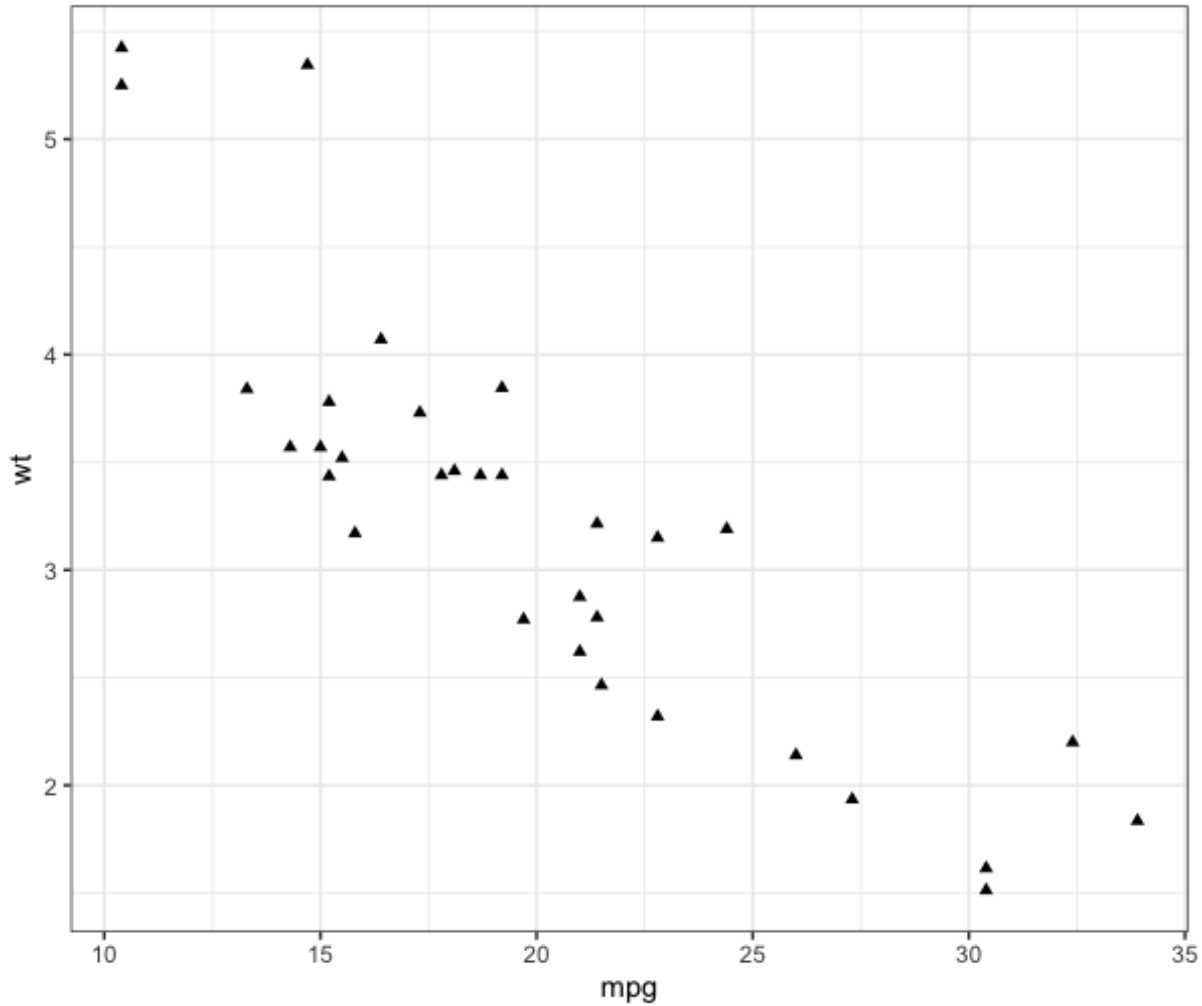
- A variety of themes exist that alter basic aspects of the visualization

- The one we have been seeing is the default, "`theme_grey`" (or "`theme_gray`", both spellings accepted



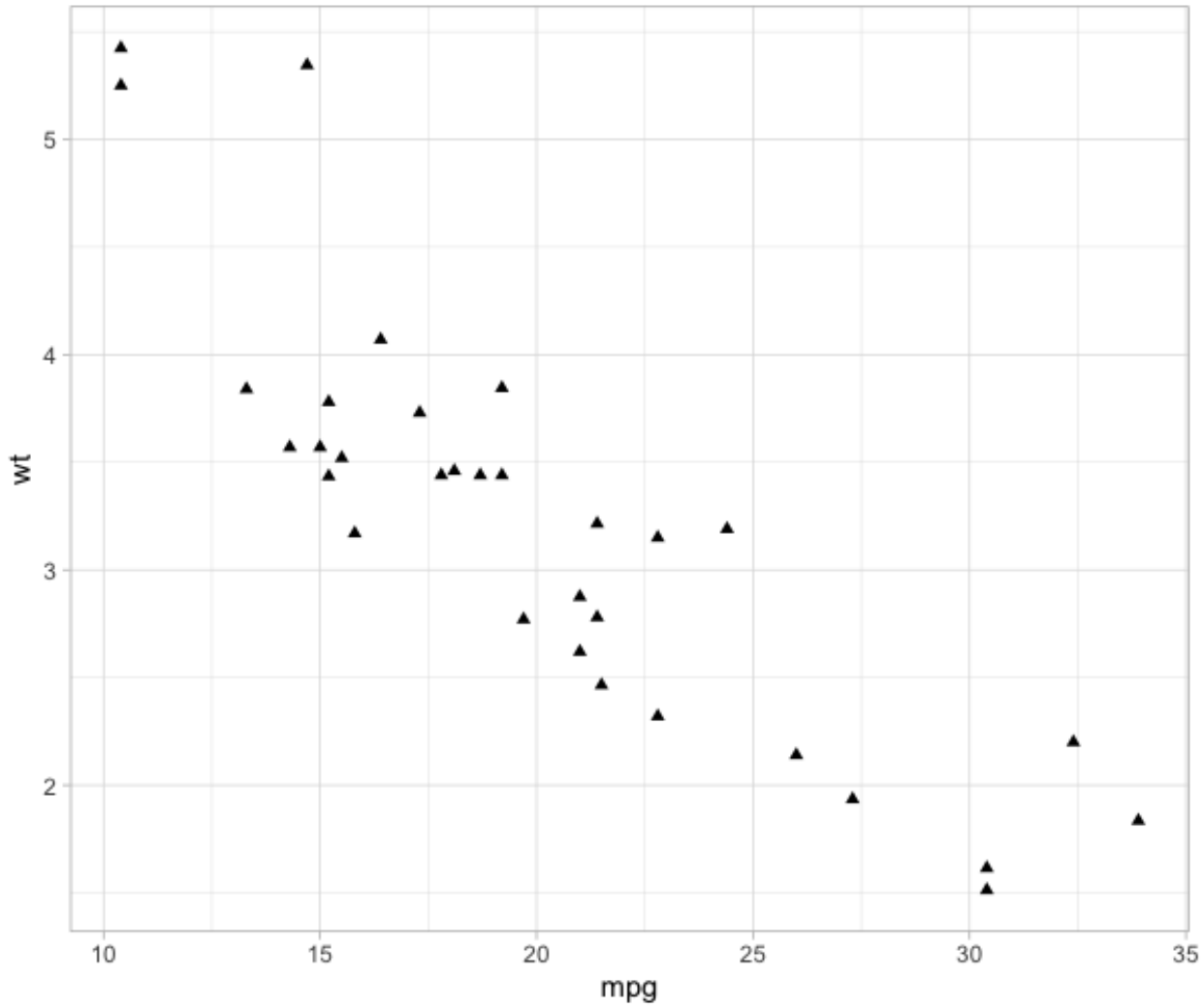
# Themes

```
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(shape=17)+
  theme_bw()
```



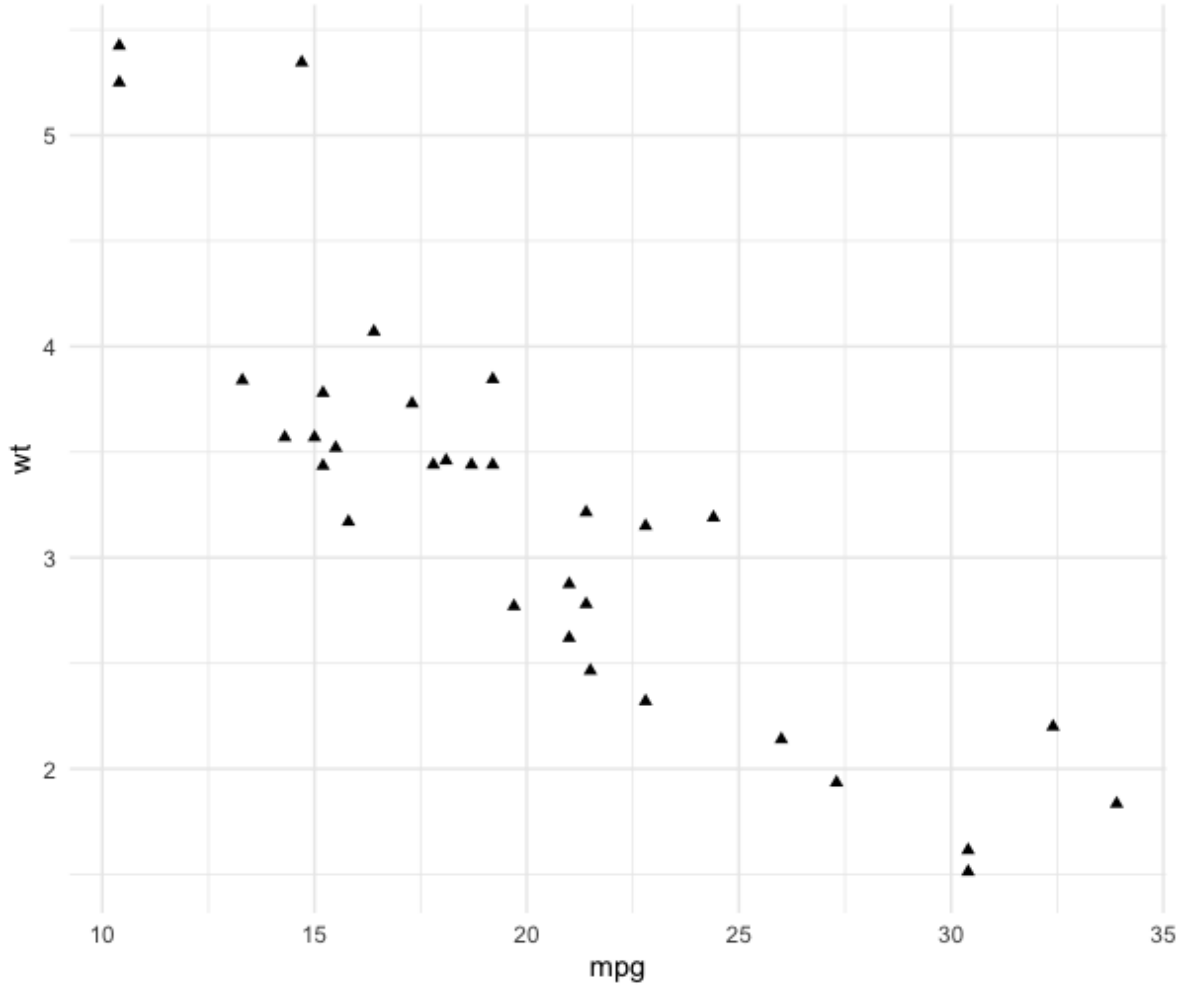
# Themes

```
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(shape=17)+
  theme_light()
```



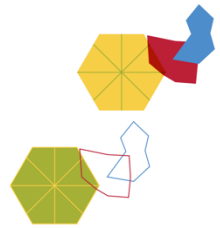
# Themes

```
ggplot(mtcars, aes(mpg, wt)) +
  geom_point(shape=17)+
  theme_minimal()
```



# Facets

- Facets allow you to plot multiple plots in separate panels in one go
- For this to work you need to have a grouping variable
- The strength of this approach over more classic ones is there is no need for loop use AND the axes are consistent
- This ensures comparability



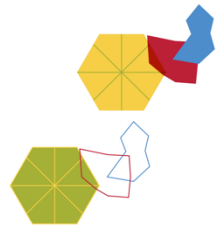
# The datasaurus dozen

```
install.packages("datasauRus")
library(datasauRus)
datasaurus_dozen
```

```
> datasaurus_dozen
# A tibble: 1,846 × 3
   dataset     x     y
   <chr>   <dbl> <dbl>
 1 dino     55.4  97.2
 2 dino     51.5  96.0
 3 dino     46.2  94.5
 4 dino     42.8  91.4
 5 dino     40.8  88.3
 6 dino     38.7  84.9
 7 dino     35.6  79.9
 8 dino     33.1  77.6
 9 dino     29.0  74.5
10 dino     26.2  71.4
# … with 1,836 more rows
# ℹ Use `print(n = ...)` to see more rows
```

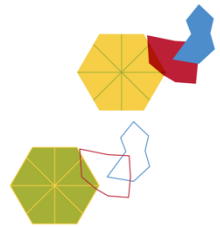
# The datasaurus dozen

```
##    dataset    mean_x mean_y std_dev_x std_dev_y corr_x_y
##    <chr>       <dbl>  <dbl>     <dbl>     <dbl>    <dbl>
##  1 away         54.3   47.8      16.8      26.9  -0.0641
##  2 bullseye     54.3   47.8      16.8      26.9  -0.0686
##  3 circle       54.3   47.8      16.8      26.9  -0.0683
##  4 dino         54.3   47.8      16.8      26.9  -0.0645
##  5 dots         54.3   47.8      16.8      26.9  -0.0603
##  6 h_lines      54.3   47.8      16.8      26.9  -0.0617
##  7 high_lines   54.3   47.8      16.8      26.9  -0.0685
##  8 slant_down   54.3   47.8      16.8      26.9  -0.0690
##  9 slant_up     54.3   47.8      16.8      26.9  -0.0686
## 10 star         54.3   47.8      16.8      26.9  -0.0630
## 11 v_lines      54.3   47.8      16.8      26.9  -0.0694
## 12 wide_lines   54.3   47.8      16.8      26.9  -0.0666
## 13 x_shape      54.3   47.8      16.8      26.9  -0.0656
```

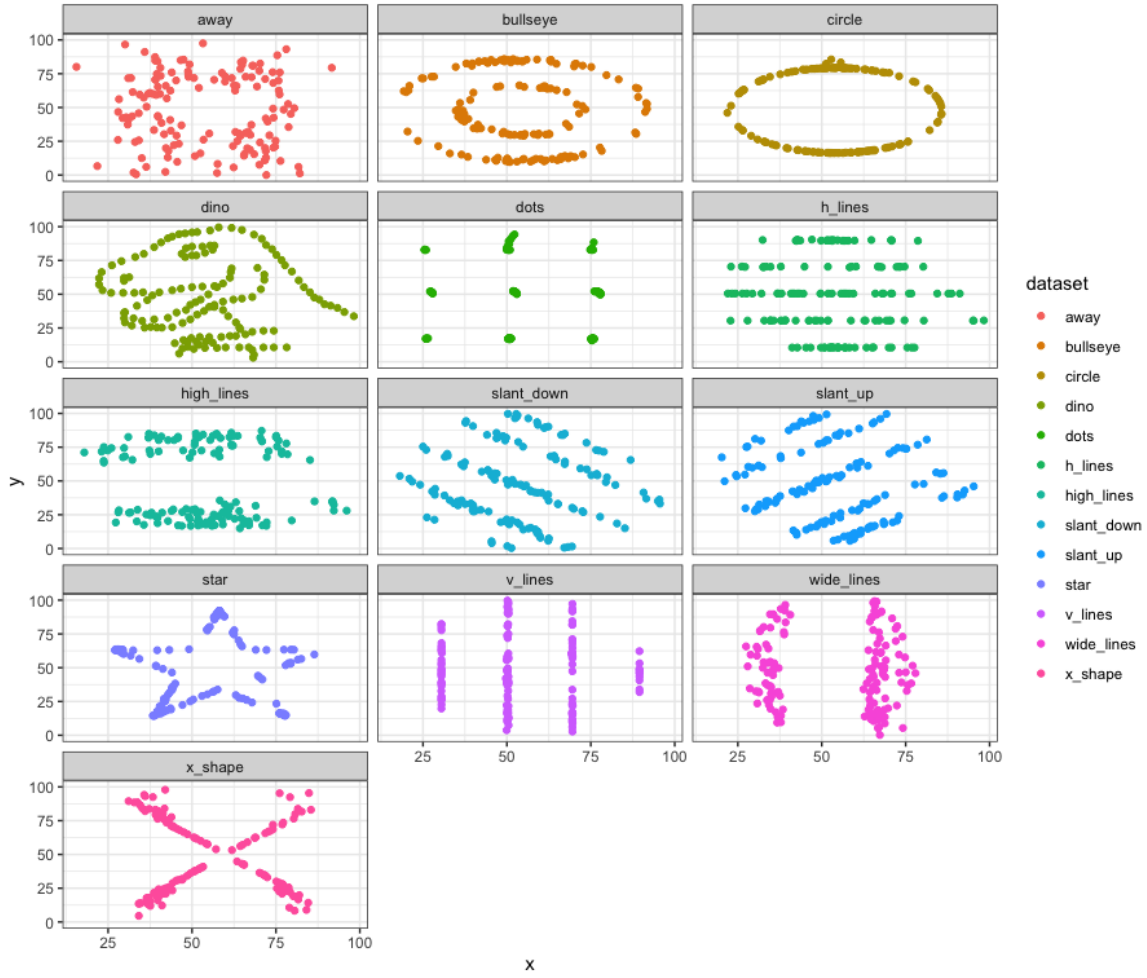


# The datasaurus dozen

```
ggplot(datasaurus_dozen, aes(x = x, y = y, colour = dataset))+
  geom_point()+
  theme_bw()+
  facet_wrap(~dataset, ncol = 3)
```

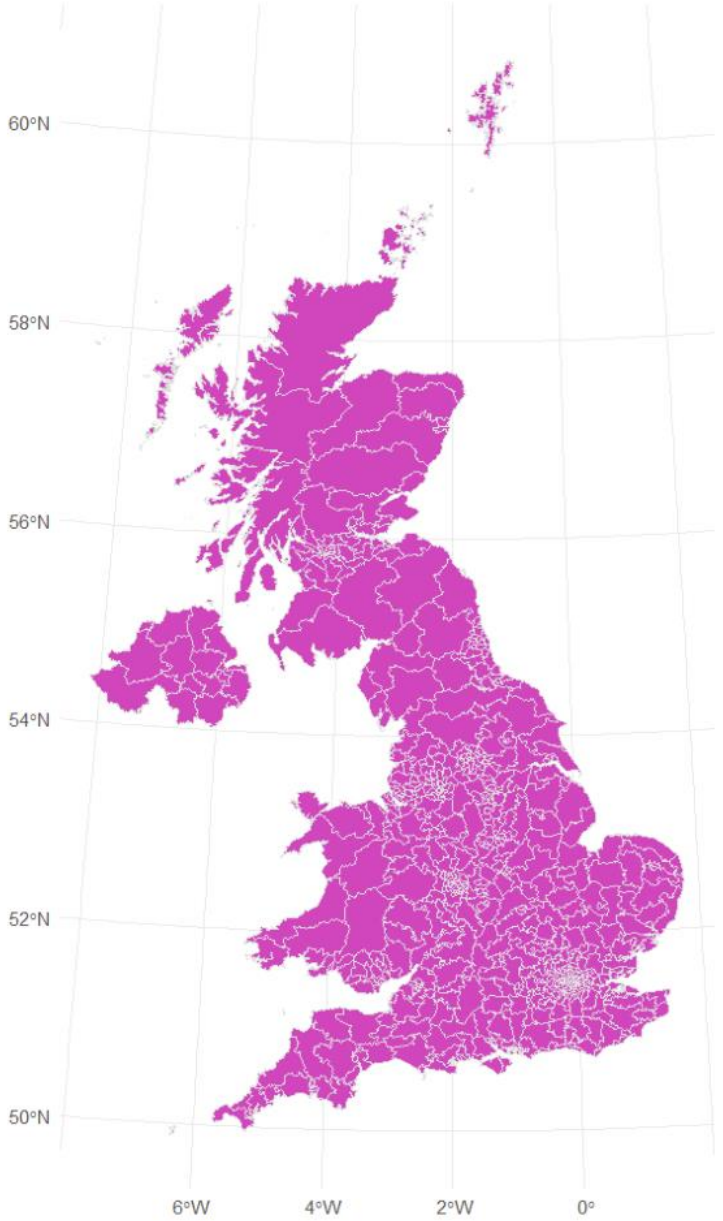


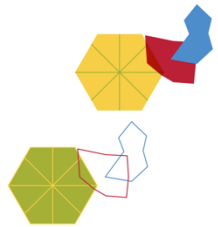
# The datasaurus dozen



# Principles of mapping

# A simple map



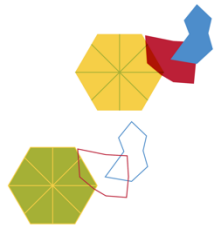



# From 3D to 2D…

- Moving from the three-dimensional entity that is planet Earth on to a flat map is very difficult!
- Consider concepts such as distances from A to B: on the flat map they are straight lines, but on a globe they are arcs
- This means that in order for maps to be accurate, we need to consider how we intend to move the 3D object into 2D
- This would involve a degree of object distortion, which is what map projections essentially do

# Projected coordinate reference systems

- Map projections have been delighting cartographers for centuries
- This has led to a myriad of map projections as they fundamentally serve different purposes
- In each situation, the projection prioritises accuracy for specific conditions (e.g., distance, or local shape)
- When a map projection (this “distortion”) and geographic coordinates (like longitude and latitude) are combined, we get a projected coordinate reference system

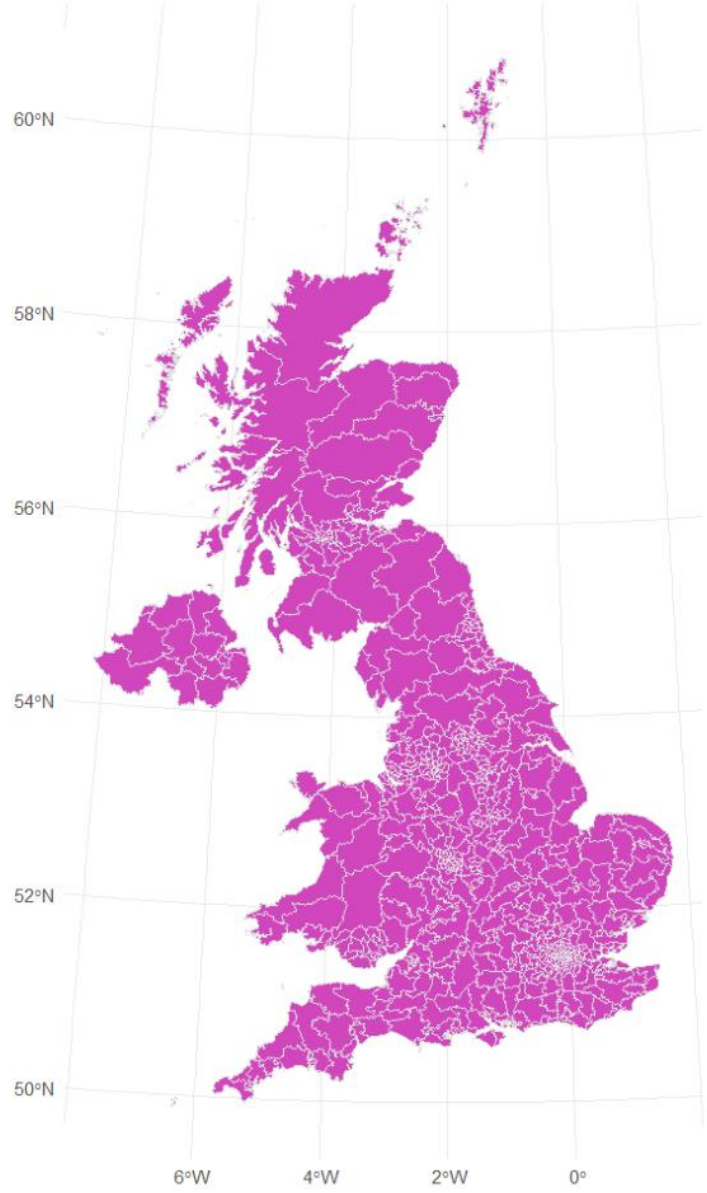


# Why do you need to know this?

- The coordinates we use for our maps would depend heavily on the Projection CRS that gave rise to them
- If we use coordinates from different systems on the same map then we will inadvertently introduce errors as these coordinates may not be directly comparable
- To shift from one coordinate system to another we need to perform a mathematical transformation

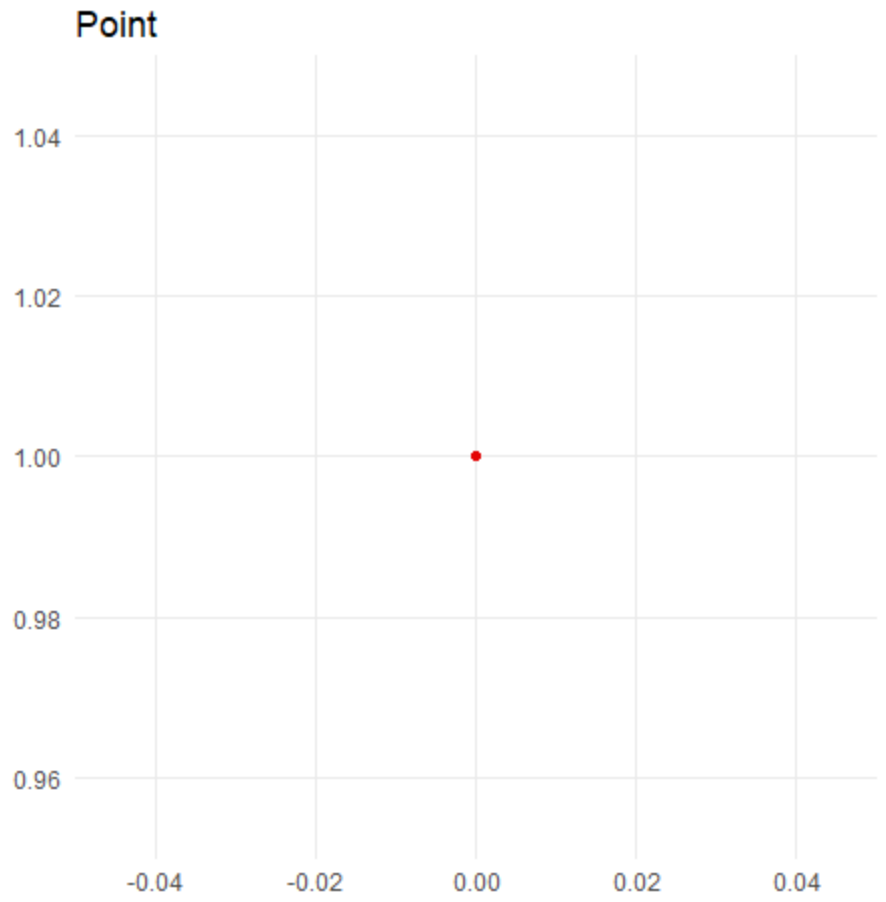
# UK constituencies

- This map has some basic components:
  - Polygons (with a grey border) – this is known as geometry
  - Longitude and latitude
  - Grid of the coordinate system



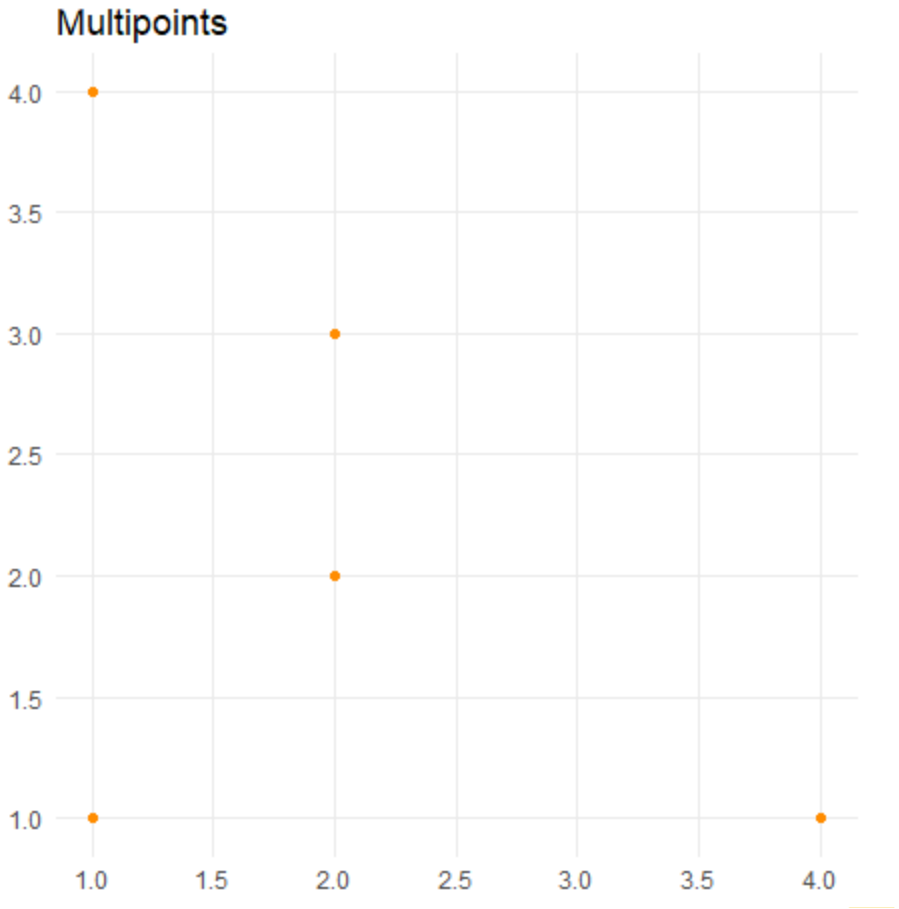
# Geometries

- There are 7 types of geometries for spatial illustrations:
  - Points



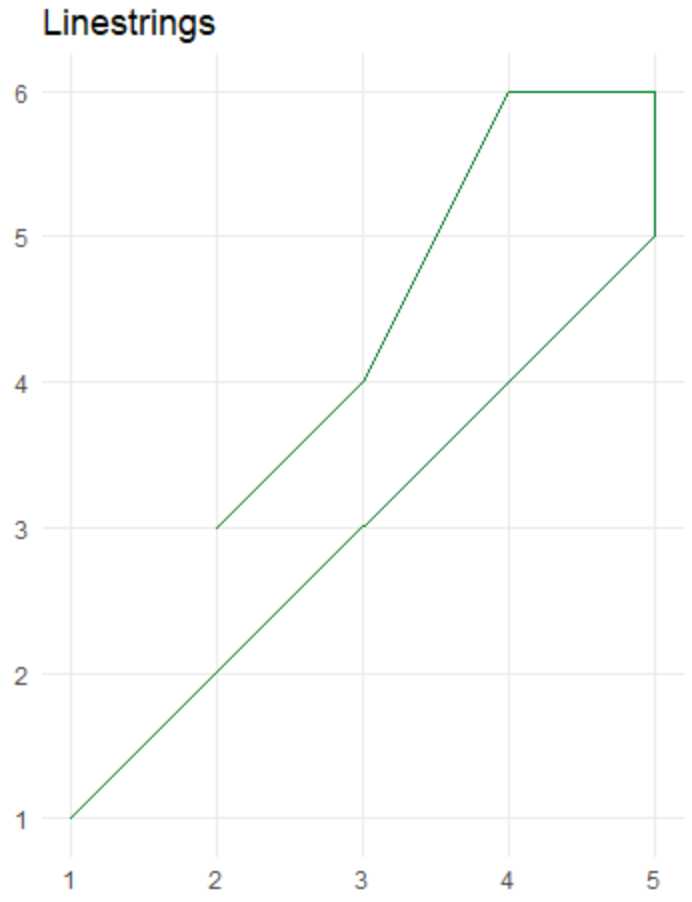
# Geometries

- There are 7 types of geometries for spatial illustrations:
  - Points
  - Multipoints



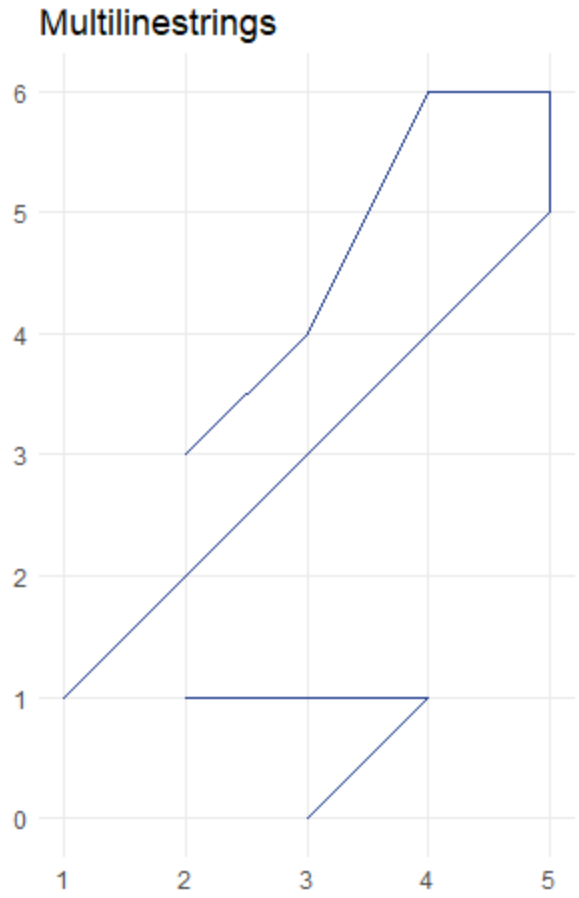
# Geometries

- There are 7 types of geometries for spatial illustrations:
  - Points
  - Multipoints
  - Linestrings



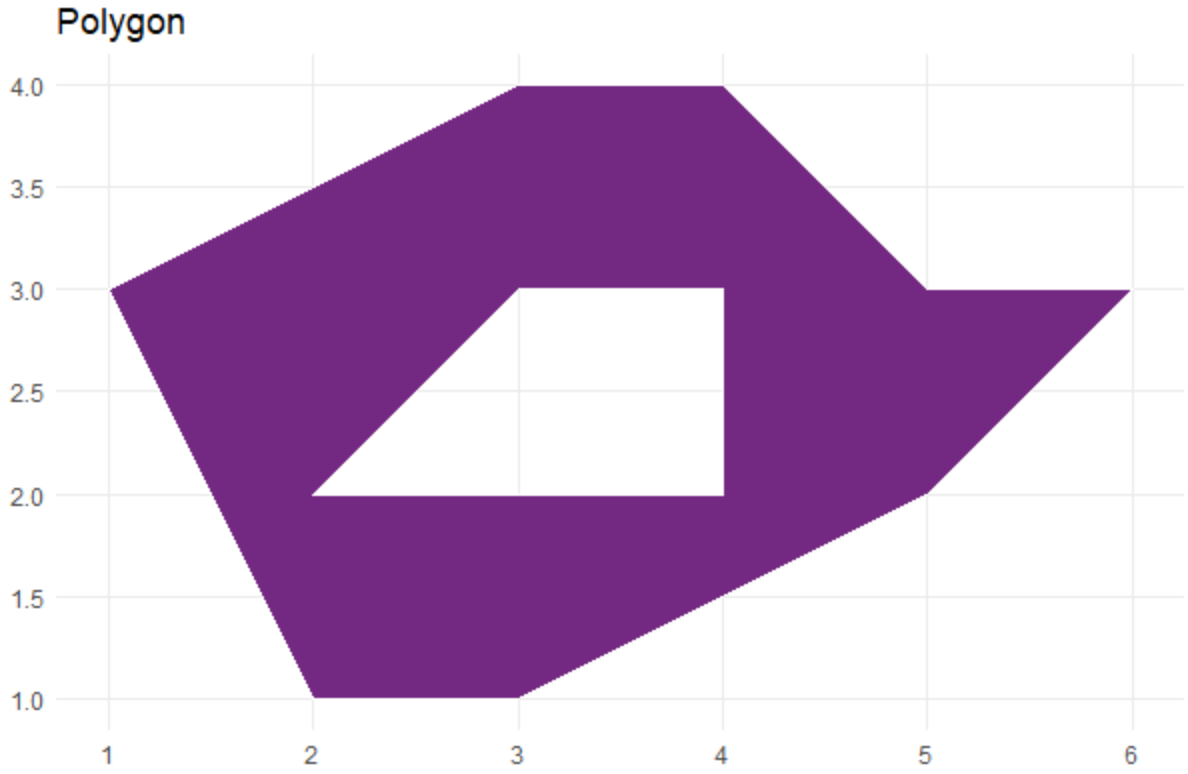
# Geometries

- There are 7 types of geometries for spatial illustrations:
  - Points
  - Multipoints
  - Linestrings
  - Multilinestrings



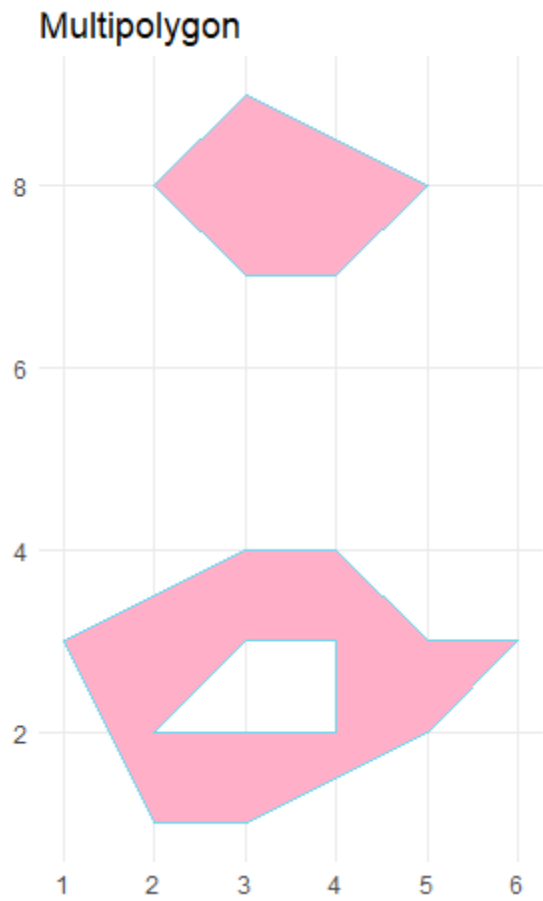
# Geometries

- There are 7 types of geometries for spatial illustrations:
  - Points
  - Multipoints
  - Linestrings
  - Multilinestrings
  - Polygons



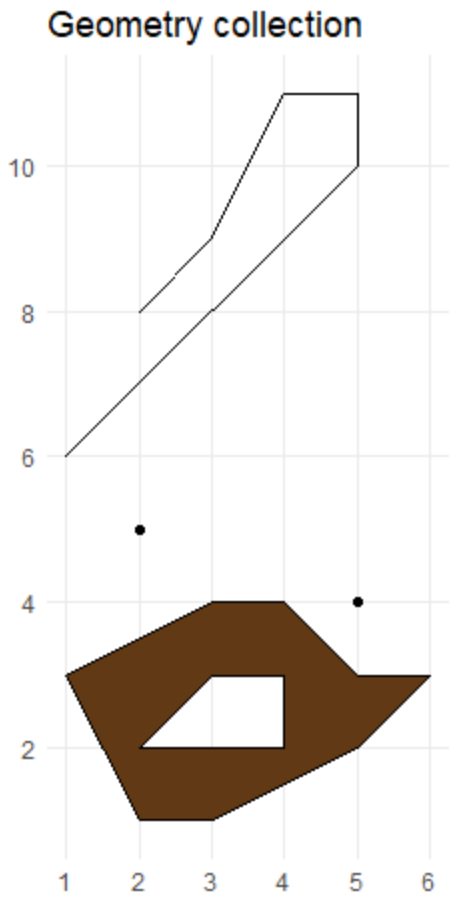
# Geometries

- There are 7 types of geometries for spatial illustrations:
  - Points
  - Multipoints
  - Linestrings
  - Multilinestrings
  - Polygons
  - Multipolygons



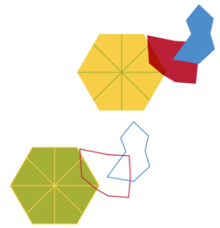
# Geometries

- There are 7 types of geometries for spatial illustrations:
  - Points
  - Multipoints
  - Linestrings
  - Multilinestrings
  - Polygons
  - Multipolygons
  - Geometry collections



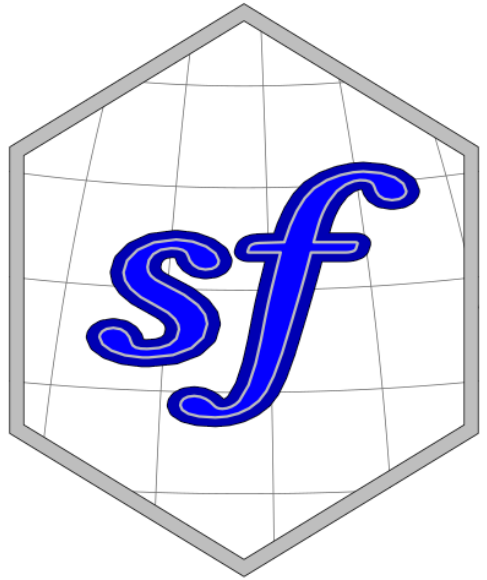
# Shape files

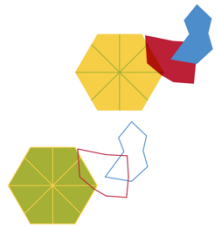
- A shape file is a collection of files that store geometry, location, and other associated information for geographic features
- For you to be able to work with shape files you need a minimum of three things:
  - .shp file: this has information on the shape and location of each feature
  - .shx file: the shape index file, it has information that allows software that reads a .shp file to quickly access a specific component in there. Think of it like a reference file
  - .dbf file: this is a database containing all the non-spatial data that are associated with each feature. Each feature has one row associated to it



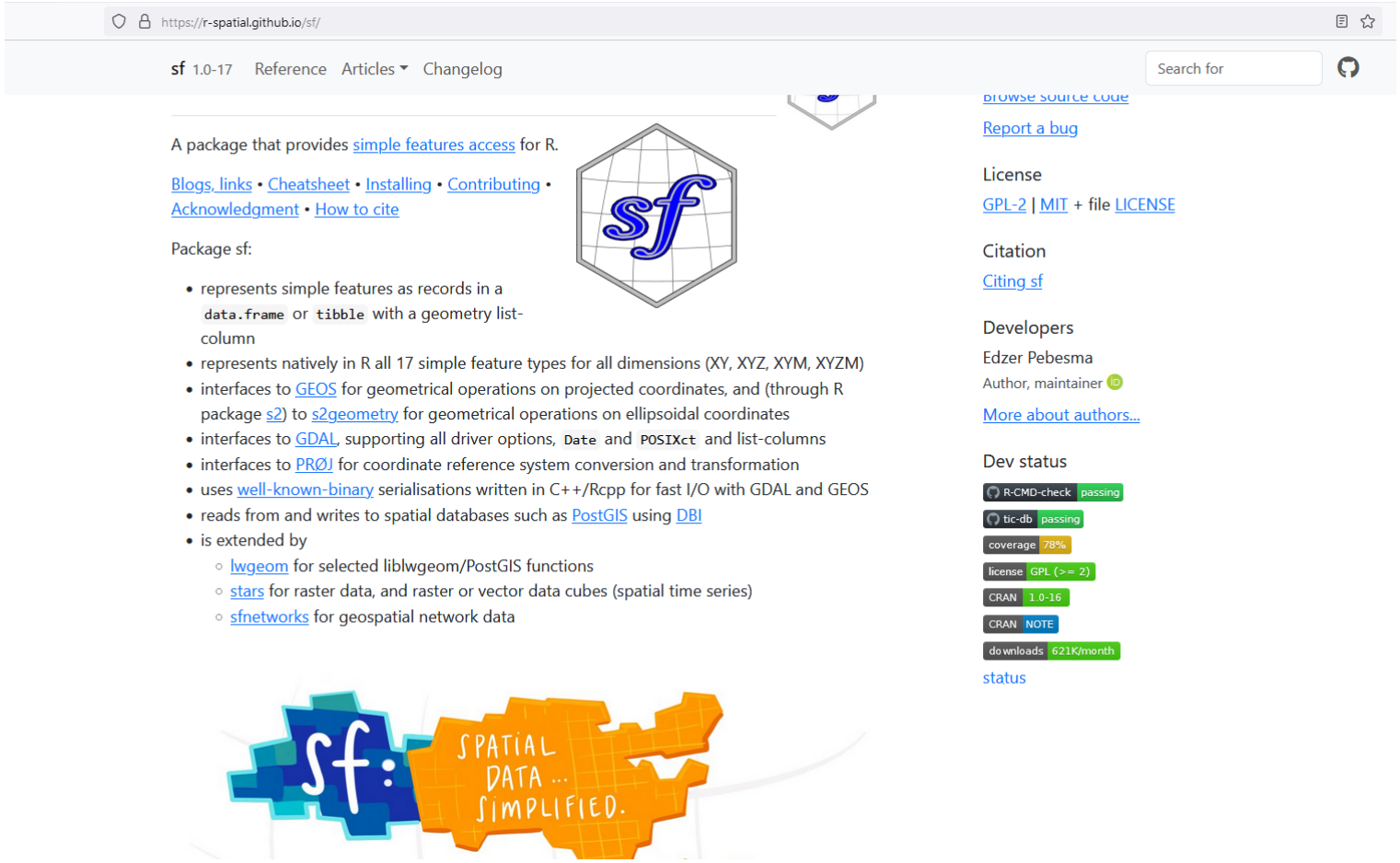
# Handling shape files in R: the sf package

- The package was developed to ensure simple feature access in R
- Essentially this means you can import, edit, and manipulate shape files
- It's fully integrated with ggplot2, ensure graphics follow consistent rules
- It is well documented with plenty of resources online
- It is extended further by other packages such as stars (for spatial time series for example)



# Where to find help: the sf package

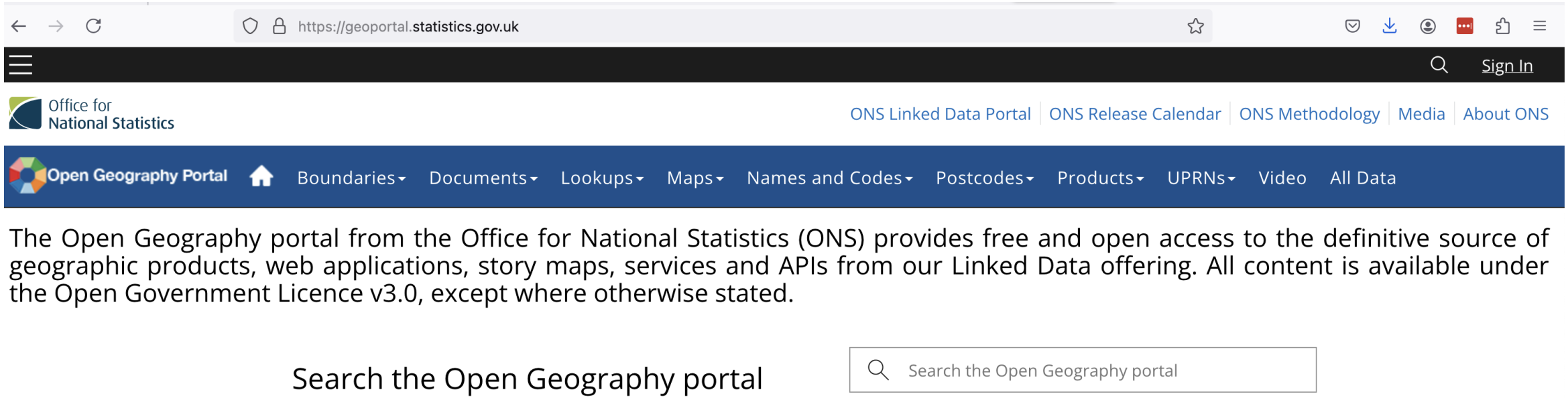


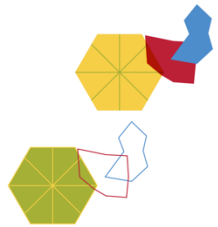
# What's in an sf object

```
> cons_sf
Simple feature collection with 650 features and 8 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -116.1487 ymin: 7054.1 xmax: 655653.8 ymax: 1220310
Projected CRS: OSGB36 / British National Grid
# A tibble: 650 × 9
   PCON24CD  PCON24NM                PCON24NMW  BNG_E  BNG_N   LONG   LAT GlobalID                           geometry
   <chr>     <chr>                   <chr>      <dbl>  <dbl>  <dbl> <dbl> <chr>                          <MULTIPOLYGON [m]>
 1 E14001063 Aldershot               NA        484716 155270 -0.786  51.3 b6eea658-887c-43bd-81a6-e879c… (((485406.9 159918.6, 48…
 2 E14001064 Aldridge-Brownhills     NA        404720 301030 -1.93   52.6 d1f29811-2c62-4dcb-b0dc-06db9… (((406519.1 305054.3, 40…
 3 E14001065 Altrincham and Sale West NA       374132 389051 -2.39   53.4 5345726c-b6b4-4d12-a310-8b2ab… (((379104.1 393143.9, 37…
 4 E14001066 Amber Valley            NA        440478 349674 -1.40   53.0 4c232f33-f30b-4526-bfec-d5aa8… (((444868.4 353958.1, 44…
 5 E14001067 Arundel and South Downs NA        497309 118530 -0.616  51.0 9eb29363-b47b-463a-9698-bc899… (((523813.2 118212, 5247…
 6 E14001068 Ashfield                NA        450035 356564 -1.25   53.1 c6664362-7ca9-43cf-9920-22201… (((455809 357962.2, 4551…
 7 E14001069 Ashford                 NA        611218 142572  1.02   51.1 d251384e-d248-457f-89da-a0b89… (((620103.1 146702.2, 61…
 8 E14001070 Ashton-under-Lyne       NA        392654 398807 -2.11   53.5 aac16076-67e4-4ab4-aae2-a413a… (((393565.7 396317.2, 39…
 9 E14001071 Aylesbury               NA        495172 216598 -0.620  51.8 975e6733-c9b0-4936-96b2-869b0… (((489373.9 224192.7, 48…
10 E14001072 Banbury                 NA        438918 232636 -1.43   52.0 36a7bbd3-7140-406a-854d-aa190… (((451341.2 245116.2, 45…
# ℹ 640 more rows
# ℹ Use `print(n = ...)` to see more rows
```

# Making simple maps

# Let’s get some data



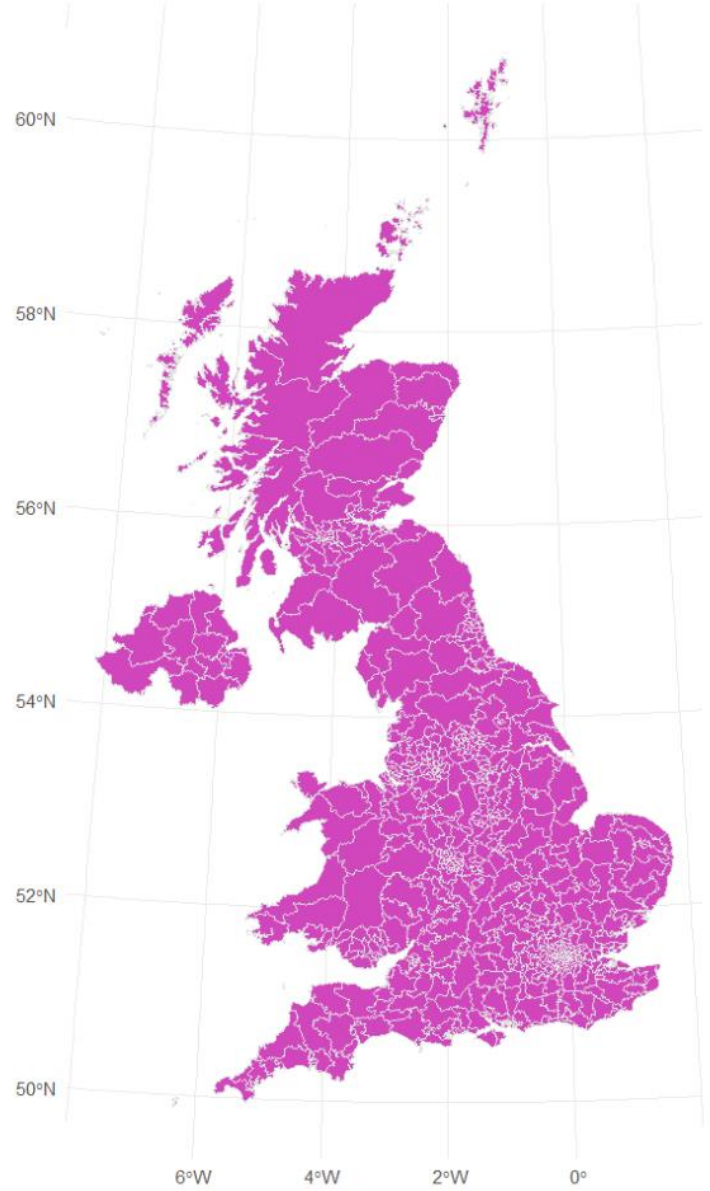



# Importing data into R

```
> library(sf)
> cons_sf <- read_sf("Data/Westminster_Parliamentary_Constituencies_July_2024_Boundaries_UK_BUC/PCON_JULY_2024_UK_BUC.shp")
> cons_sf
Simple feature collection with 650 features and 8 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -116.1487 ymin: 7054.1 xmax: 655653.8 ymax: 1220310
Projected CRS: OSGB36 / British National Grid
# A tibble: 650 × 9
   PCON24CD  PCON24NM                 PCON24NMW  BNG_E  BNG_N   LONG   LAT GlobalID                                geometry
   <chr>     <chr>                    <chr>      <dbl>  <dbl>  <dbl> <dbl> <chr>                                <MULTIPOLYGON [m]>
 1 E14001063 Aldershot                NA        484716 155270 -0.786  51.3 b6eea658-887c-43bd-81a6-e879c… (((485406.9 159918.6, 48…
 2 E14001064 Aldridge-Brownhills      NA        404720 301030 -1.93   52.6 d1f29811-2c62-4dcb-b0dc-06db9… (((406519.1 305054.3, 40…
 3 E14001065 Altrincham and Sale West NA        374132 389051 -2.39   53.4 5345726c-b6b4-4d12-a310-8b2ab… (((379104.1 393143.9, 37…
 4 E14001066 Amber Valley             NA        440478 349674 -1.40   53.0 4c232f33-f30b-4526-bfec-d5aa8… (((444868.4 353958.1, 44…
 5 E14001067 Arundel and South Downs  NA        497309 118530 -0.616  51.0 9eb29363-b47b-463a-9698-bc899… (((523813.2 118212, 5247…
 6 E14001068 Ashfield                 NA        450035 356564 -1.25   53.1 c6664362-7ca9-43cf-9920-22201… (((455809 357962.2, 4551…
 7 E14001069 Ashford                  NA        611218 142572  1.02   51.1 d251384e-d248-457f-89da-a0b89… (((620103.1 146702.2, 61…
 8 E14001070 Ashton-under-Lyne        NA        392654 398807 -2.11   53.5 aac16076-67e4-4ab4-aae2-a413a… (((393565.7 396317.2, 39…
 9 E14001071 Aylesbury                NA        495172 216598 -0.620  51.8 975e6733-c9b0-4936-96b2-869b0… (((489373.9 224192.7, 48…
10 E14001072 Banbury                  NA        438918 232636 -1.43   52.0 36a7bbd3-7140-406a-854d-aa190… (((451341.2 245116.2, 45…
# ℹ 640 more rows
# ℹ Use `print(n = ...)` to see more rows
```

# Let us start plotting

```
ggplot(cons_sf) +
  geom_sf(fill = "#d946bb", color = "grey88") +
  theme_minimal()
```

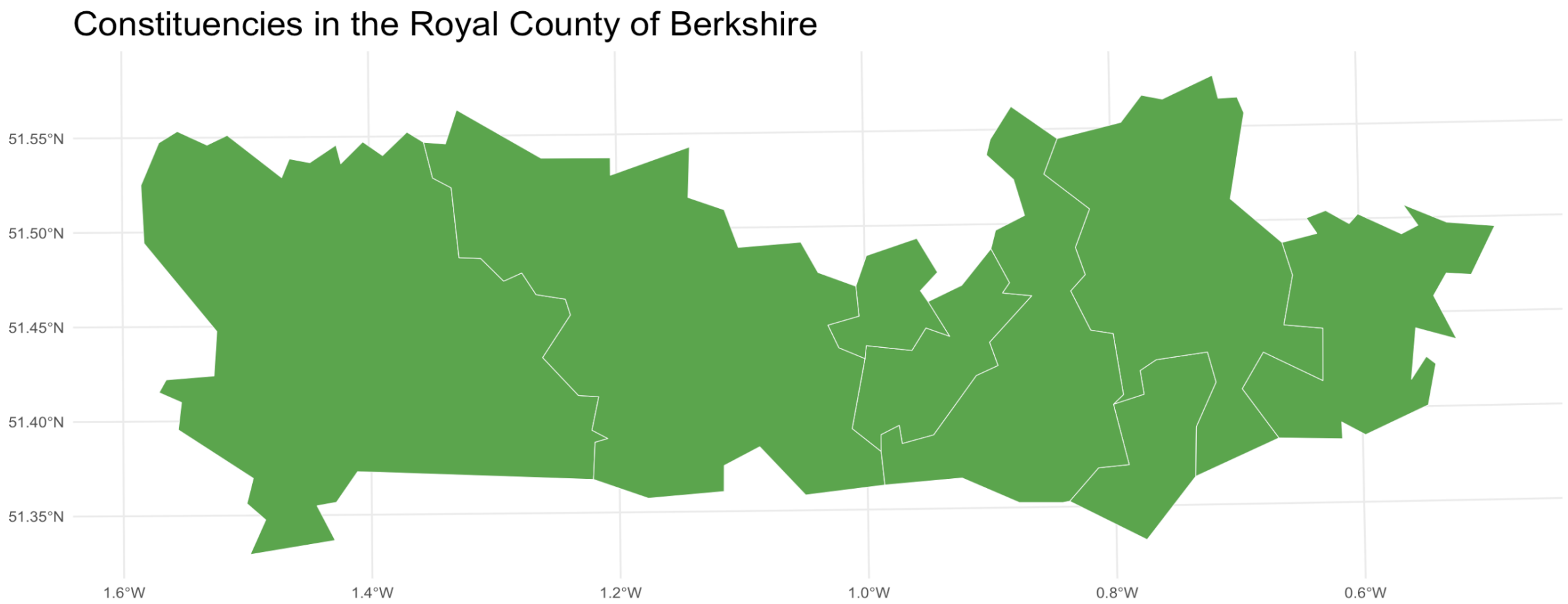


# Focusing on Berkshire

```
cons_sf %>%
  filter(PCON24NM %in% c("Bracknell","Maidenhead","Newbury",
                         "Reading West and Mid Berkshire",
                         "Windsor","Wokingham","Bracknell", "Maidenhead",
                         "Reading Central","Earley and Woodley"))->berkshire_sf
```

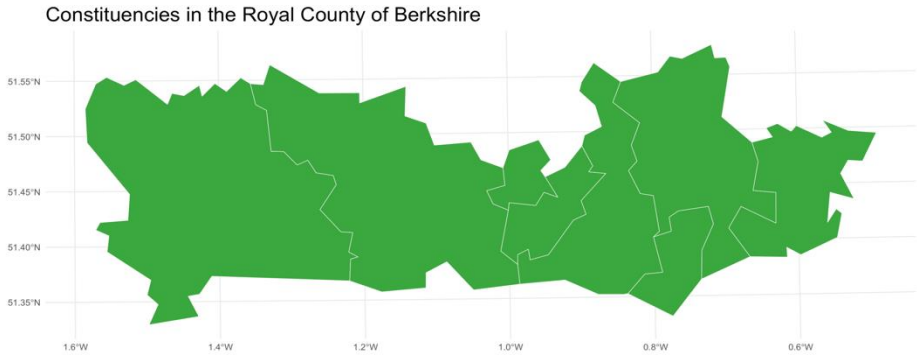
```
Simple feature collection with 8 features and 8 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 428908.2 ymin: 159037.9 xmax: 504919.2 ymax: 187236.2
Projected CRS: OSGB36 / British National Grid
# A tibble: 8 × 9
  PCON24CD  PCON24NM                      PCON24NMW  BNG_E  BNG_N   LONG   LAT GlobalID                          geometry
* <chr>     <chr>                         <chr>      <dbl>  <dbl>  <dbl> <dbl> <chr>                       <MULTIPOLYGON [m]>
1 E14001117 Bracknell                     NA        485168 164319 -0.778  51.4 0cedda70-1085-4b97-a3fe-… (((488139.3 163641.5, 48…
2 E14001210 Earley and Woodley            NA        473892 169821 -0.939  51.4 a1e7bf67-92e9-4ef0-818a-… (((470474.7 165097.6, 46…
3 E14001348 Maidenhead                    NA        487442 178744 -0.742  51.5 13b67738-4d88-4681-b1bc-… (((492975.3 177400.9, 49…
4 E14001376 Newbury                       NA        441905 173263 -1.40   51.5 5d192482-9900-4c04-848a-… (((444774.3 183291.8, 44…
5 E14001438 Reading Central               NA        471018 173955 -0.979  51.5 367a3b3f-7579-49cb-ab75-… (((473102.2 173916.3, 47…
6 E14001439 Reading West and Mid Berkshire NA       457864 172341 -1.17   51.4 e5c3976a-498e-4d1a-812b-… (((459709.4 183018, 4596…
7 E14001588 Windsor                       NA        497834 172353 -0.594  51.4 e486c083-786c-44c9-a6a9-… (((502226 178595.4, 5049…
8 E14001593 Wokingham                     NA        477342 167203 -0.890  51.4 83b16614-5cd7-4a55-b838-… (((480348.5 183509.3, 47…
```

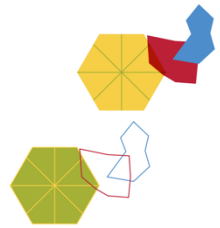
```r
ggplot(berkshire_sf) +
  geom_sf(fill = "#3BA740", color = "white")+
  labs(title="Constituencies in the Royal County of Berkshire")+
  theme_minimal()+
  theme(plot.title=element_text(size=20))
```



# Adding labels

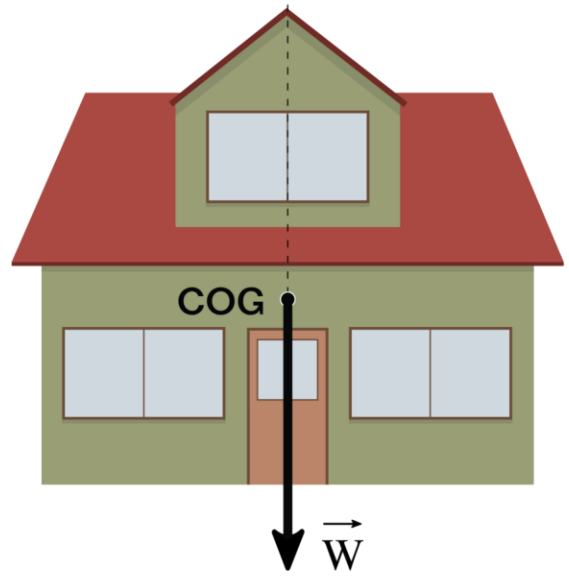
- As useful as a base map is, without labels it is difficult to communicate clearly what the reader should be gaining out of it
- In this map, we could label what each constituency is called
- This brings us into the next difficulty: Where do you position each label?

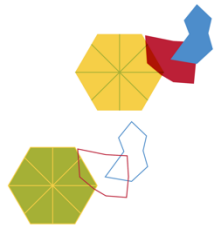


# Finding a middle ground..

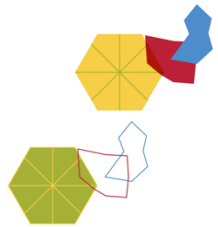
- It’s easy to find the geometric centre in a regularly shaped object, but with irregular objects it gets trickier
- Enter the concept of “centroid”
- It is essentially what in physics would be the centre of gravity, and the calculation is rather involved
- Thankfully, there are R functions to do this on our behalf



# Finding a middle ground…

```
> berkshire_sf$centroid <- st_centroid(berkshire_sf$geometry)
> berkshire_sf$centroid
Geometry set for 8 features
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 440821.6 ymin: 165424 xmax: 497511 ymax: 176484.4
Projected CRS: OSGB36 / British National Grid
First 5 geometries:
POINT (485770.9 165424)
POINT (473563 170495)
POINT (487381.4 176484.4)
POINT (440821.6 172200.6)
POINT (471000.5 173894.2)
> centroids <- st_coordinates(berkshire_sf$centroid)
> centroids
            X        Y
[1,] 485770.9 165424.0
[2,] 473563.0 170495.0
[3,] 487381.4 176484.4
[4,] 440821.6 172200.6
[5,] 471000.5 173894.2
[6,] 458476.2 172982.4
[7,] 497511.0 172718.5
[8,] 478811.3 170604.2
> berkshire_sf <- cbind(berkshire_sf, centroids)
```



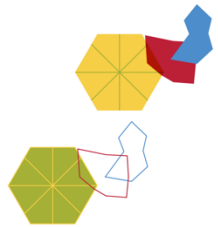
# Finding a middle ground..

```
> berkshire_st
Simple feature collection with 8 features and 10 fields
Active geometry column: geometry
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 428908.2 ymin: 159037.9 xmax: 504919.2 ymax: 187236.2
Projected CRS: OSGB36 / British National Grid
   PCON24CD                      PCON24NM PCON24NMW  BNG_E  BNG_N     LONG     LAT                              GlobalID
1 E14001117                     Bracknell      <NA> 485168 164319 -0.77784 51.3716 0cedda70-1085-4b97-a3fe-0f634dfd688a
2 E14001210           Earley and Woodley      <NA> 473892 169821 -0.93865 51.4226 a1e7bf67-92e9-4ef0-818a-132aecb94778
3 E14001348                   Maidenhead      <NA> 487442 178744 -0.74162 51.5009 13b67738-4d88-4681-b1bc-ff5ecea8a91b
4 E14001376                      Newbury      <NA> 441905 173263 -1.39825 51.4568 5d192482-9900-4c04-848a-af390ec00918
5 E14001438                Reading Central      <NA> 471018 173955 -0.97915 51.4601 367a3b3f-7579-49cb-ab75-e539e243d939
6 E14001439 Reading West and Mid Berkshire      <NA> 457864 172341 -1.16872 51.4471 e5c3976a-498e-4d1a-812b-d9380b9a3198
7 E14001588                      Windsor      <NA> 497834 172353 -0.59372 51.4417 e486c083-786c-44c9-a6a9-759eb52e0ed5
8 E14001593                    Wokingham      <NA> 477342 167203 -0.88961 51.3986 83b16614-5cd7-4a55-b838-f7c308733746
         X        Y                       geometry                    centroid
1 485770.9 165424.0 MULTIPOLYGON (((488139.3 16...   POINT (485770.9 165424)
2 473563.0 170495.0 MULTIPOLYGON (((470474.7 16...     POINT (473563 170495)
3 487381.4 176484.4 MULTIPOLYGON (((492975.3 17... POINT (487381.4 176484.4)
4 440821.6 172200.6 MULTIPOLYGON (((444774.3 18... POINT (440821.6 172200.6)
5 471000.5 173894.2 MULTIPOLYGON (((473102.2 17... POINT (471000.5 173894.2)
6 458476.2 172982.4 MULTIPOLYGON (((459709.4 18... POINT (458476.2 172982.4)
7 497511.0 172718.5 MULTIPOLYGON (((502226 1785...   POINT (497511 172718.5)
8 478811.3 170604.2 MULTIPOLYGON (((480348.5 18... POINT (478811.3 170604.2)
```

# Adding labels

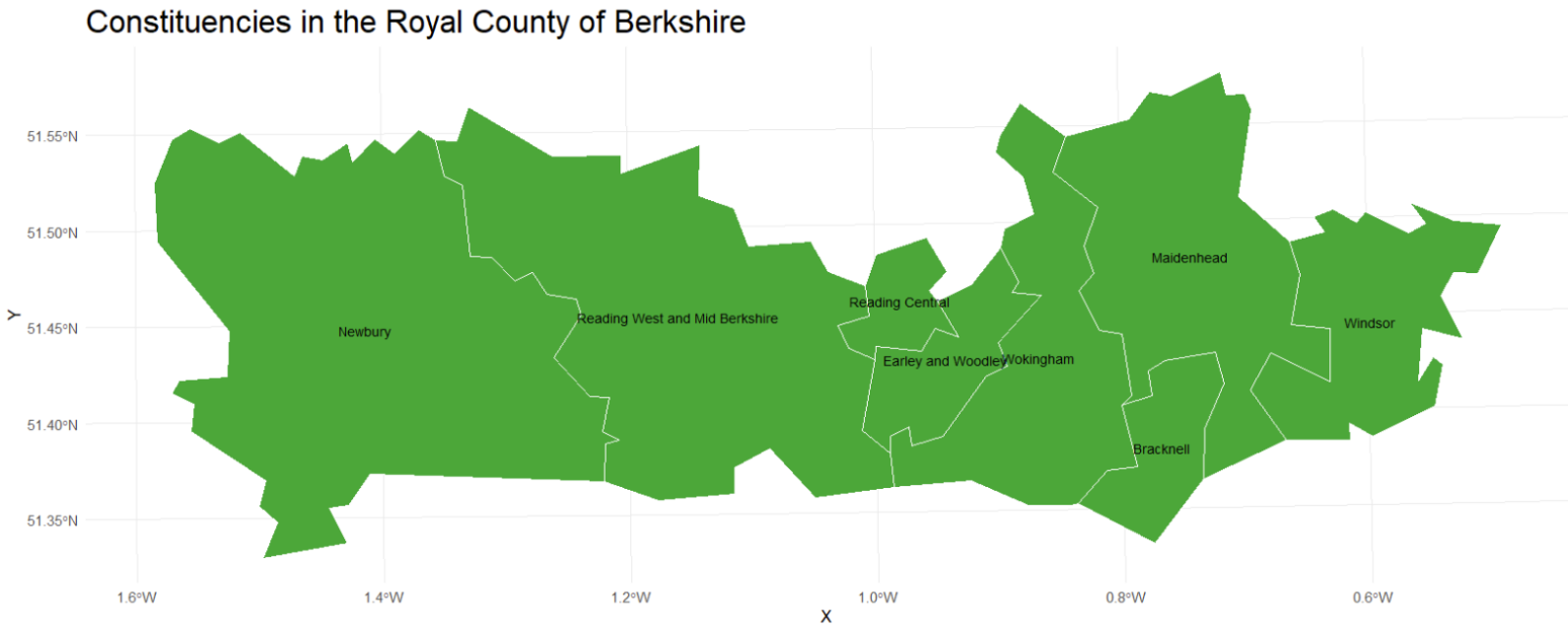
- We now have the location for each point, and we can get the name of each constituency from the tibble

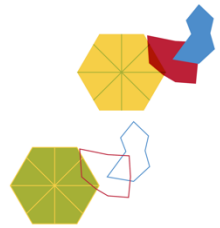
```
ggplot(berkshire_sf) +
  geom_sf(fill = "#3BA740", color = "white")+
  labs(title="Constituencies in the Royal County of Berkshire")+
  geom_text(aes(x = X, y = Y, label = PCON24NM), size = 3, color = "black") +
  theme_minimal()+
  theme(plot.title=element_text(size=20))
```



# Adding labels

- We now have the location for each point, and we can get the name of each constituency from the tibble

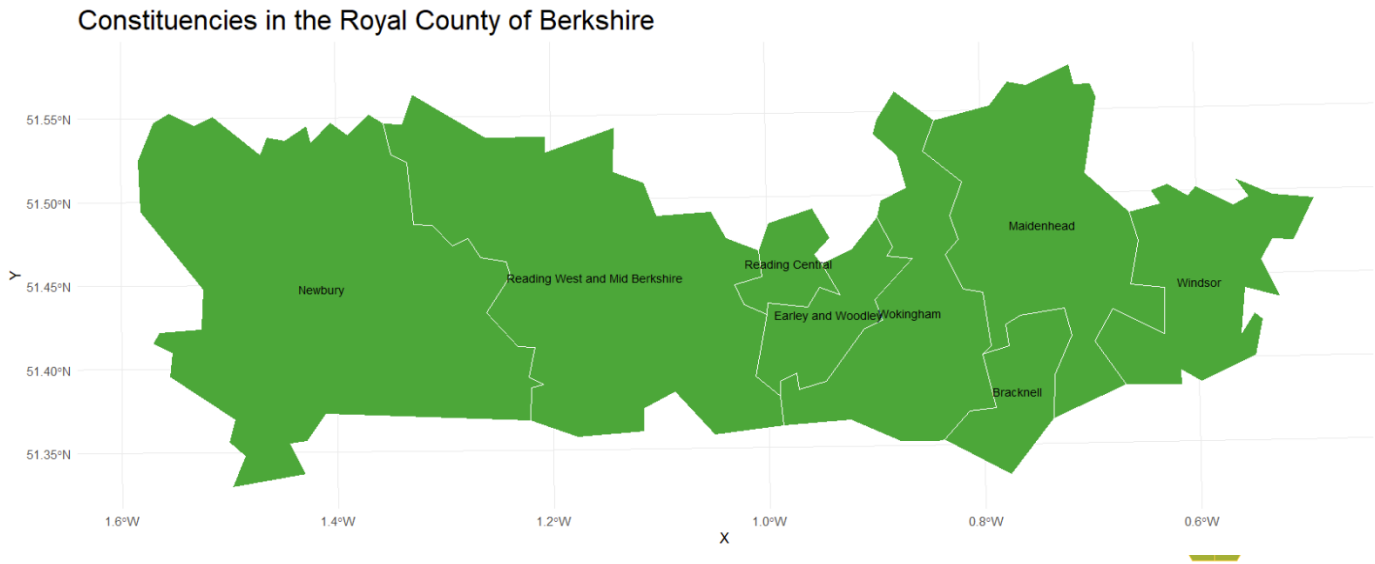


# Adding labels

- We now have the location for each point, and we can get the name of each constituencv from the tibble

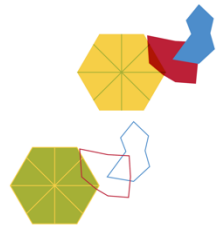
```
ggplot(berkshire_sf) +
  geom_sf(fill = "#3BA740", color = "white")+
  labs(title="Constituencies in the Royal County of Berkshire")+
  geom_text(aes(x = X, y = Y, label = PCON24NM), size = 3, color = "black") +
  theme_minimal()+
  theme(plot.title=element_text(size=20))
```



# A little bit of data handling…

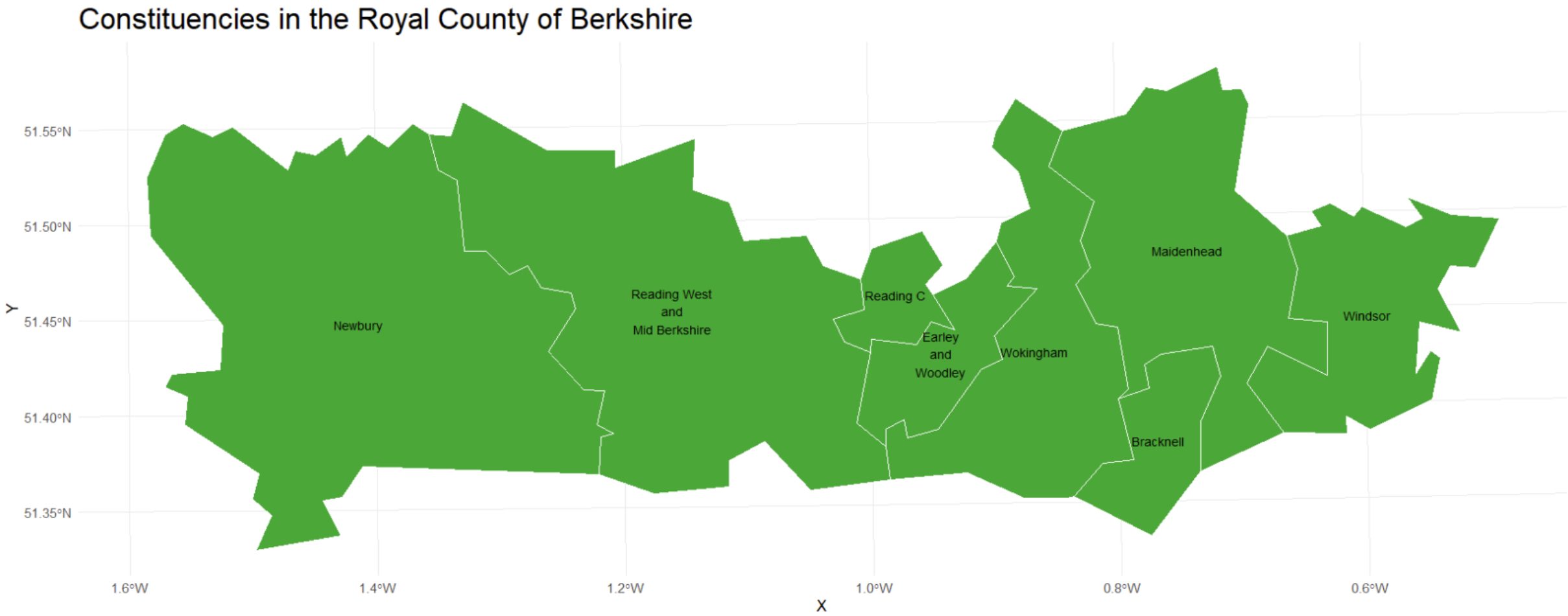
```
berkshire_sf %<>%
  mutate(name = case_when(PCON24NM=="Reading West and Mid Berkshire"~"Reading West\nand\nMid Berkshire",
                          PCON24NM=="Reading Central"~"Reading C",
                          PCON24NM=="Earley and Woodley"~"Earley\nand\nWoodley",
                          TRUE~PCON24NM))
```

```
berkshire_sf %>% select(name)

Simple feature collection with 8 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 428908.2 ymin: 159037.9 xmax: 504919.2 ymax: 187236.2
Projected CRS: OSGB36 / British National Grid
                            name                       geometry
1                       Bracknell MULTIPOLYGON (((488139.3 16...
2              Earley\nand\nWoodley MULTIPOLYGON (((470474.7 16...
3                       Maidenhead MULTIPOLYGON (((492975.3 17...
4                          Newbury MULTIPOLYGON (((444774.3 18...
5                       Reading C MULTIPOLYGON (((473102.2 17...
6 Reading West\nand\nMid Berkshire MULTIPOLYGON (((459709.4 18...
7                          Windsor MULTIPOLYGON (((502226 1785...
8                       Wokingham MULTIPOLYGON (((480348.5 18...
```



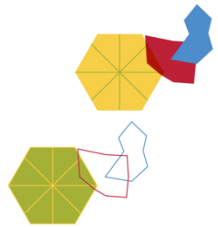
# Adding labels

```
ggplot(berkshire_sf) +
  geom_sf(fill = "#3BA740", color = "white")+
  labs(title="Constituencies in the Royal County of Berkshire")+
  geom_text(aes(x = X, y = Y, label = name), size = 3, color = "black") +
  theme_minimal()+
  theme(plot.title=element_text(size=20))
```



# Layering over some colour

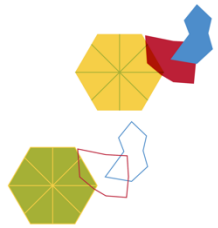
- Often we may want to represent different characteristics for each polygon using colour
- The two option available are to use discrete colours or continuous
  - Discrete colours are for variables that are categorical
  - Continuous ones are for variables that can take any values within an interval



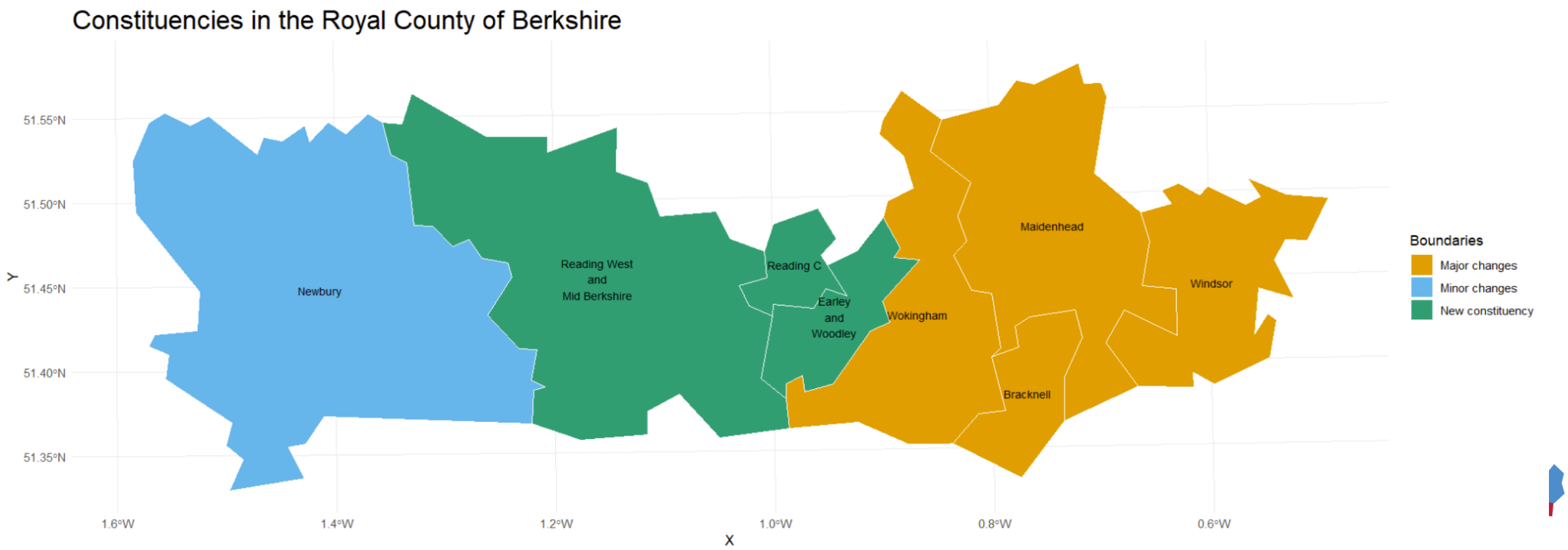
# Layering over some colour

- In our constituencies in Berkshire map we can create a variable that describes whether the constituency is completely new, remained unchanged in the latest re-organisation or had some changes

```
Boundaries<-c("Major changes","New constituency", "Major changes",
              "Minor changes","New constituency","New constituency",
              "Major changes", "Major changes")

berkshire_sf<- cbind(berkshire_sf, Boundaries)

ggplot(berkshire_sf) +
  geom_sf(aes(fill = Boundaries), color = "white")+
  scale_fill_manual(values=c("#E69F00", "#56B4E9","#009E73"))+
  labs(title="Constituencies in the Royal County of Berkshire")+
  geom_text(aes(x = X, y = Y, label = name), size = 3, color = "black") +
  theme_minimal()+
  theme(plot.title=element_text(size=20))
```



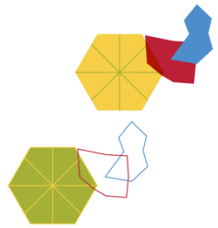
# Layering over some colour



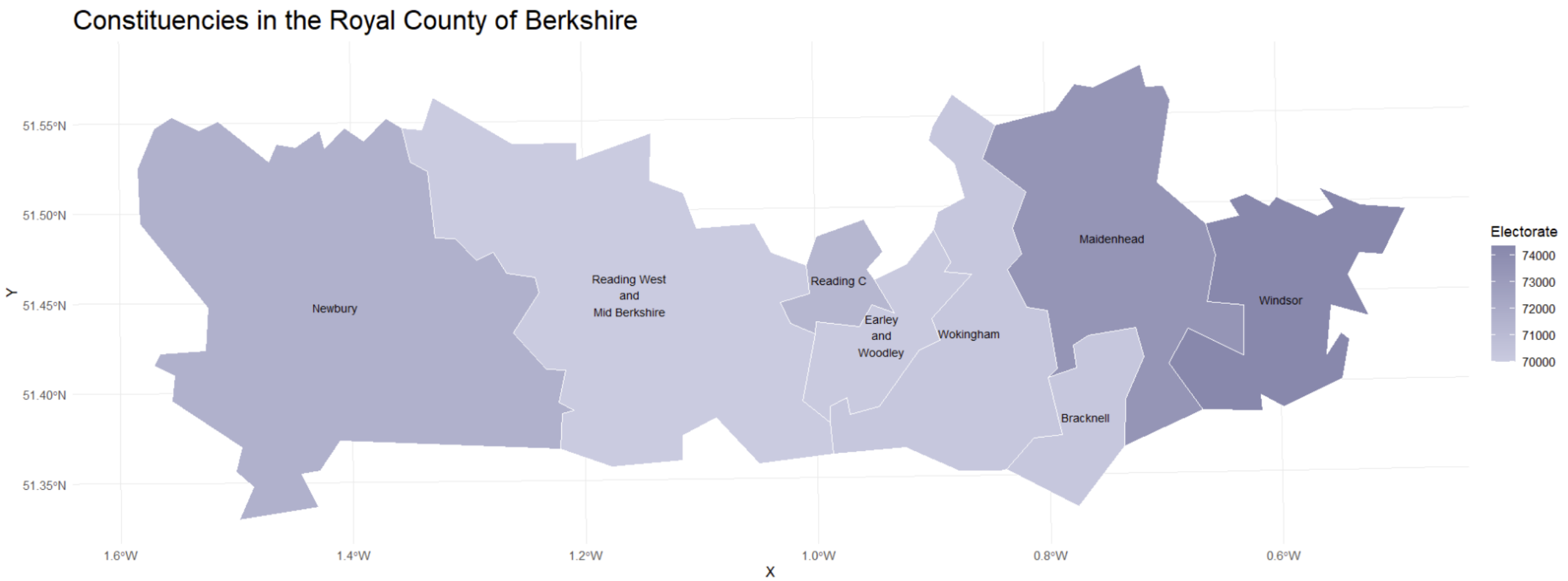
# Layering over some colour

- For a continuous variable we can include the electorate size of each constituency

```
Electorate<-c(70247,70083,73463,71631,71283,69999,74338,70235)

berkshire_sf<- cbind(berkshire_sf, Electorate)

ggplot(berkshire_sf) +
  geom_sf(aes(fill = Electorate), color = "white") +
  scale_fill_continuous(name = "Electorate", high = "#8788ab", low = "#cacbdf")+
  geom_text(aes(x = X, y = Y, label = name), size = 3, color = "black") +
  labs(title = "Constituencies in the Royal County of Berkshire") +
  theme_minimal() +
  theme(plot.title = element_text(size = 20))
```

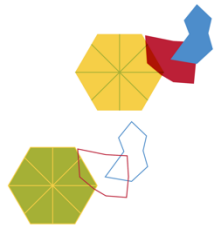


# Layering over some colour



# Adding points

- We can also add specific points of interest on our maps
- The important aspect is to ensure that we have translated our new points to the coordinate system of our base map
- Let us say that we want:
  - To include the 8 largest settlements in those constituencies
  - Using the longitude and latitudes in degrees for the centre of each of these
- How do we do that?



# Adding points

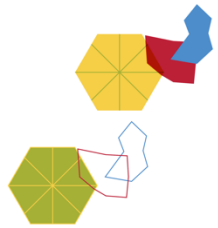
- Step 1: Create a dataframe of longitude and latitudes

```
Settlements <- data.frame(
  Town =c("Reading", "Bracknell", "Maidenhead","Wokingham", "Newbury",
          "Woodley", "Thatcham","Sandhurst" ),
  lat = c(51.4551,51.4141,51.5218,51.410,51.4014, 51.4517, 51.4058, 51.3462),
  lon = c(-0.9787,-0.7526,-0.7242,-0.8339,-1.3231, -0.9029, -1.2665,-0.8043)
)
```

Description: df [8 × 3]

| Town <chr> | lat <dbl> | lon <dbl> |
|---|---|---|
| Reading | 51.4551 | -0.9787 |
| Bracknell | 51.4141 | -0.7526 |
| Maidenhead | 51.5218 | -0.7242 |
| Wokingham | 51.4100 | -0.8339 |
| Newbury | 51.4014 | -1.3231 |
| Woodley | 51.4517 | -0.9029 |
| Thatcham | 51.4058 | -1.2665 |
| Sandhurst | 51.3462 | -0.8043 |

8 rows

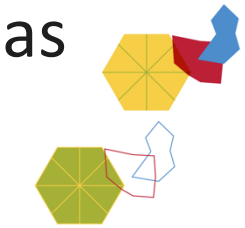


# Adding points

- Step 2: Turn it into an sf object

```
> Settlements_sf <- st_as_sf(Settlements, coords = c("lon", "lat"), crs = 4326)
> Settlements_sf
Simple feature collection with 8 features and 1 field
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -1.3231 ymin: 51.3462 xmax: -0.7242 ymax: 51.5218
Geodetic CRS:  WGS 84
        Town                  geometry
1    Reading POINT (-0.9787 51.4551)
2  Bracknell POINT (-0.7526 51.4141)
3 Maidenhead POINT (-0.7242 51.5218)
4  Wokingham   POINT (-0.8339 51.41)
5    Newbury POINT (-1.3231 51.4014)
6    Woodley POINT (-0.9029 51.4517)
7   Thatcham POINT (-1.2665 51.4058)
8  Sandhurst POINT (-0.8043 51.3462)
```

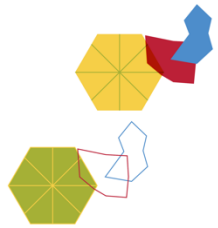
- “crs 4326” essentially states that these are coordinates presented as longitude and latitude on the surface of the Earth



# Adding points

- Step 3: Convert them to the same crs as our base map

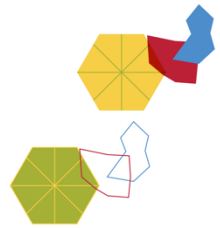
```
> Settlements_sf <- st_transform(Settlements_sf, st_crs(berkshire_sf))
> Settlements_sf
Simple feature collection with 8 features and 1 field
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 447184.5 ymin: 161469.2 xmax: 488612.1 ymax: 181091.5
Projected CRS: OSGB36 / British National Grid
        Town                    geometry
1    Reading POINT (471058.7 173396.6)
2  Bracknell POINT (486845.7 169080.2)
3 Maidenhead POINT (488612.1 181091.5)
4  Wokingham POINT (481199.3 168531.2)
5    Newbury POINT (447184.5 167147.1)
6    Woodley   POINT (476331 173094.6)
7   Thatcham POINT (451116.9 167674.3)
8  Sandhurst POINT (483373.6 161469.2)
```



# Adding points

- Step 4: For ease of use, create a coordinates table

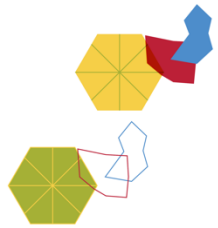
```
> settlements_coords <- st_coordinates(Settlements_sf)
>
> settlements_coords
             X        Y
[1,] 471058.7 173396.6
[2,] 486845.7 169080.2
[3,] 488612.1 181091.5
[4,] 481199.3 168531.2
[5,] 447184.5 167147.1
[6,] 476331.0 173094.6
[7,] 451116.9 167674.3
[8,] 483373.6 161469.2
```



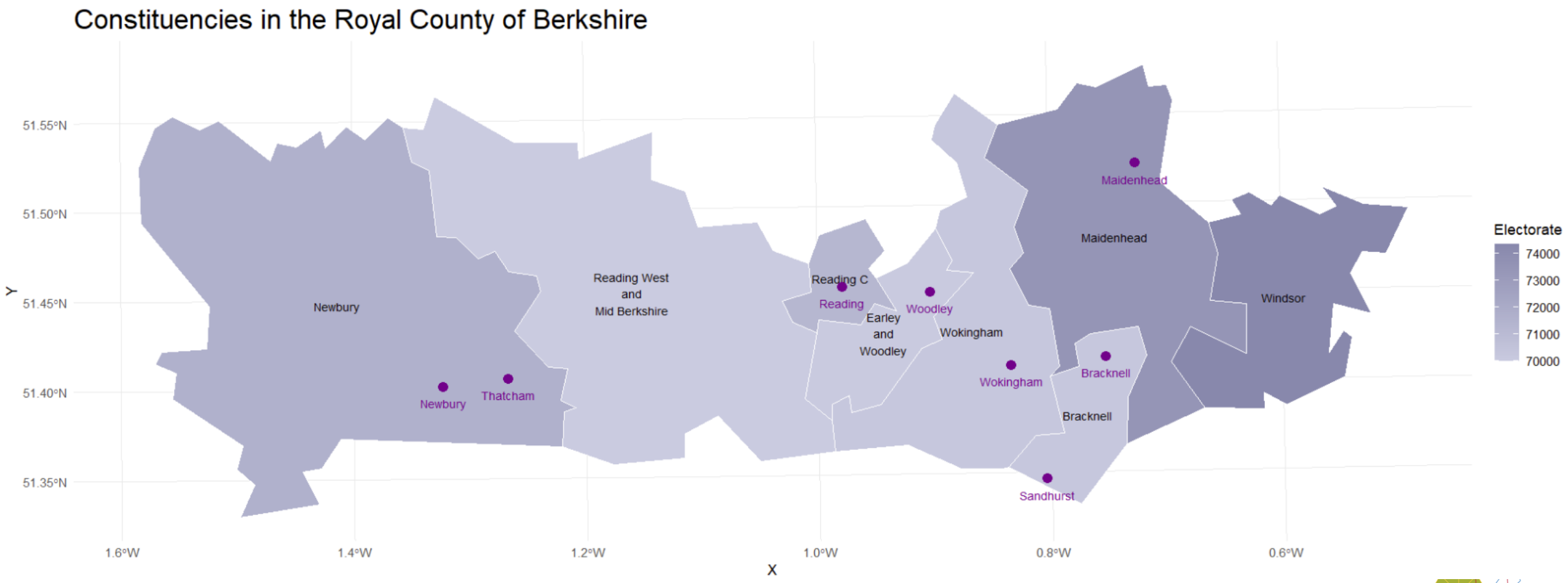
# Adding points

- Step 5: Use `geom_point` and `geom_text` on `ggplot` to add your points

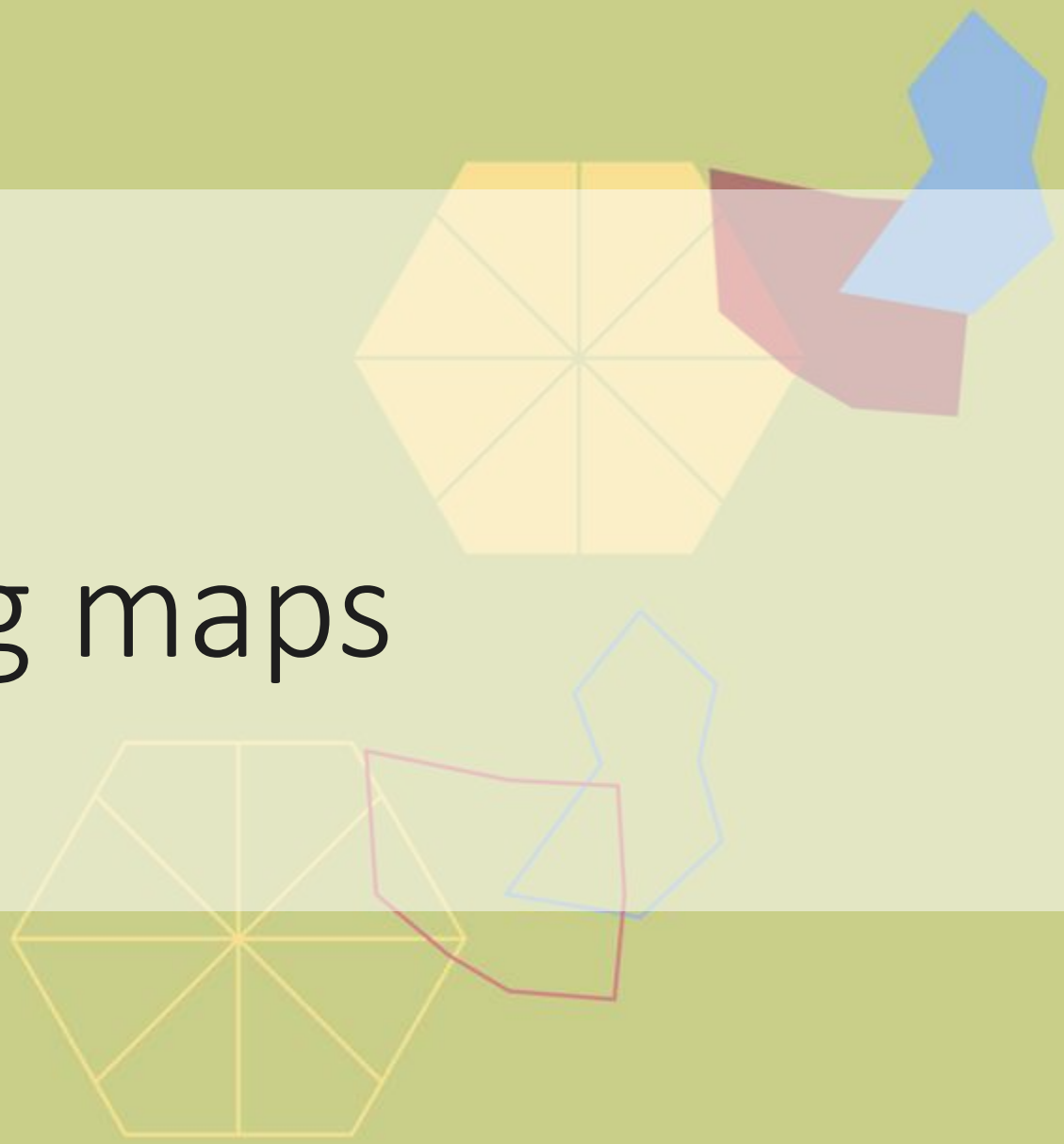
```
ggplot(berkshire_sf) +
  geom_sf(aes(fill = Electorate), color = "white") +
  scale_fill_continuous(name = "Electorate", high = "#8788ab", low = "#cacbdf")+
  geom_text(aes(x = X, y = Y, label = name), size = 3, color = "black") +
  geom_point(data = Settlements_sf,
             aes(x = settlements_coords[,1], y = settlements_coords[,2]),
             color = "#770088", size = 3,pch=19) +
  geom_text(data = Settlements_sf,
            aes(x = settlements_coords[,1], y = settlements_coords[,2], label=Town),
            size = 3, nudge_y = -1000, color = "#770088") +
  labs(title = "Constituencies in the Royal County of Berkshire") +
  theme_minimal() +
  theme(plot.title = element_text(size = 20))
```



# Adding points

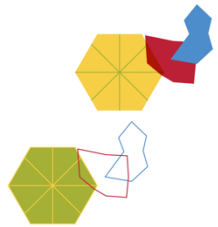


# Layering maps



# What about layering maps?

- In the same way as adding points to our map, we may want to add polygons
- This can be useful if we want to, for example, indicate the distribution of a particular species or a specific habitat
- Within the `ggplot` setup, this is actually rather straightforward



# Let us use an example

https://publications.naturalengland.org.uk/publication/4918342350798848

NATURAL ENGLAND

Access to Evidence

Home Map Additional information Feedback

Search

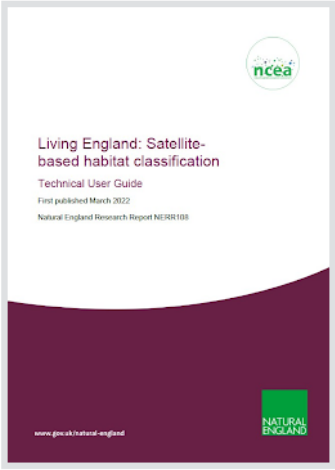
Home

## Living England: Satellite-based habitat classification- Technical User Guide (NERR108)

This record was published by Natural England on 31 March 2022.

Natural England Research Reports

Living England project, led by Natural England, is a multi-year programme delivering a satellite-derived national habitat layer in support of the Environmental Land Management (ELM) System and the Natural Capital and Ecosystem Assessment (NCEA) Pilot. The project uses a machine learning approach to image classification, developed under the Defra Living Maps project (SD1705 – Kilcoyne et al., 2017). The method first clusters homogeneous areas of habitat into segments, then assigns each segment to a defined list of habitat classes using Random Forest (a machine learning algorithm). The habitat probability map displays modelled likely broad habitat classifications, trained on field surveys and earth observation data from 2021 as well as historic data layers. This map is an output from Phase IV of the Living England project, with future work in Phase V (2022-23) intending to standardise the methodology and Phase VI (2023-24) intending to produce a baseline version using the agreed standardised methods.



# Accessing the data – The 14 zones

NATURAL ENGLAND

Accessibility | Accessing Data | Web Apps | Conservation Strategy | Privacy and Cookies | Contact Us | More

## Biogeographic Zones Living England (2021)

Authoritative

**Natural England Open Data Publication**
Defra group ArcGIS Online organisation

View Map | Download | More

### Summary

Biogeographic Zones Living England (2021) - This is a spatial dataset which defines the regions for the Living England Phase IV habitat classification.

This is a spatial dataset which defines the regions for the Living England Phase IV habitat classification.

14 regions were created in England to balance resource requirements and scalability. The regions are based on National Character Areas which are grouped such that each region is covered by a single European Space Agency Sentinel-2 satellite orbit (with the exception of Zone 10 in the SE which is covered by two orbits), and such that the regions are approximately similar in size.

For more information about the Living England Habitat Map see the Living England Satellite-based habitat classification - Technical User Guide (NERR108): publications.naturalengland.org.uk/publication/4918342350798848

### Details

**Dataset**
Feature Layer

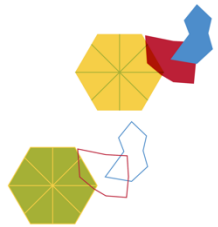
**6 June 2023**
Info Updated

**25 April 2022**
Data Updated

**13 July 2021**
Published Date

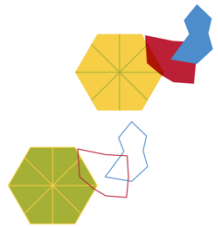
**Records: 14**
View data table

**Public**
Anyone can see this content



# First let us make a map of England…

- In this example we want to layer the zones over a base map
  - In theory the data from Natural England do not require a base map but for illustration purposes we are going to layer the zones over a map of England
- We can easily make a map of England from our constituency dataset by excluding Scottish, Welsh, and Northern Irish constituencies
- Our dataset, helpfully, has a constituency code under variable PCON24CD
- English constituencies start with E

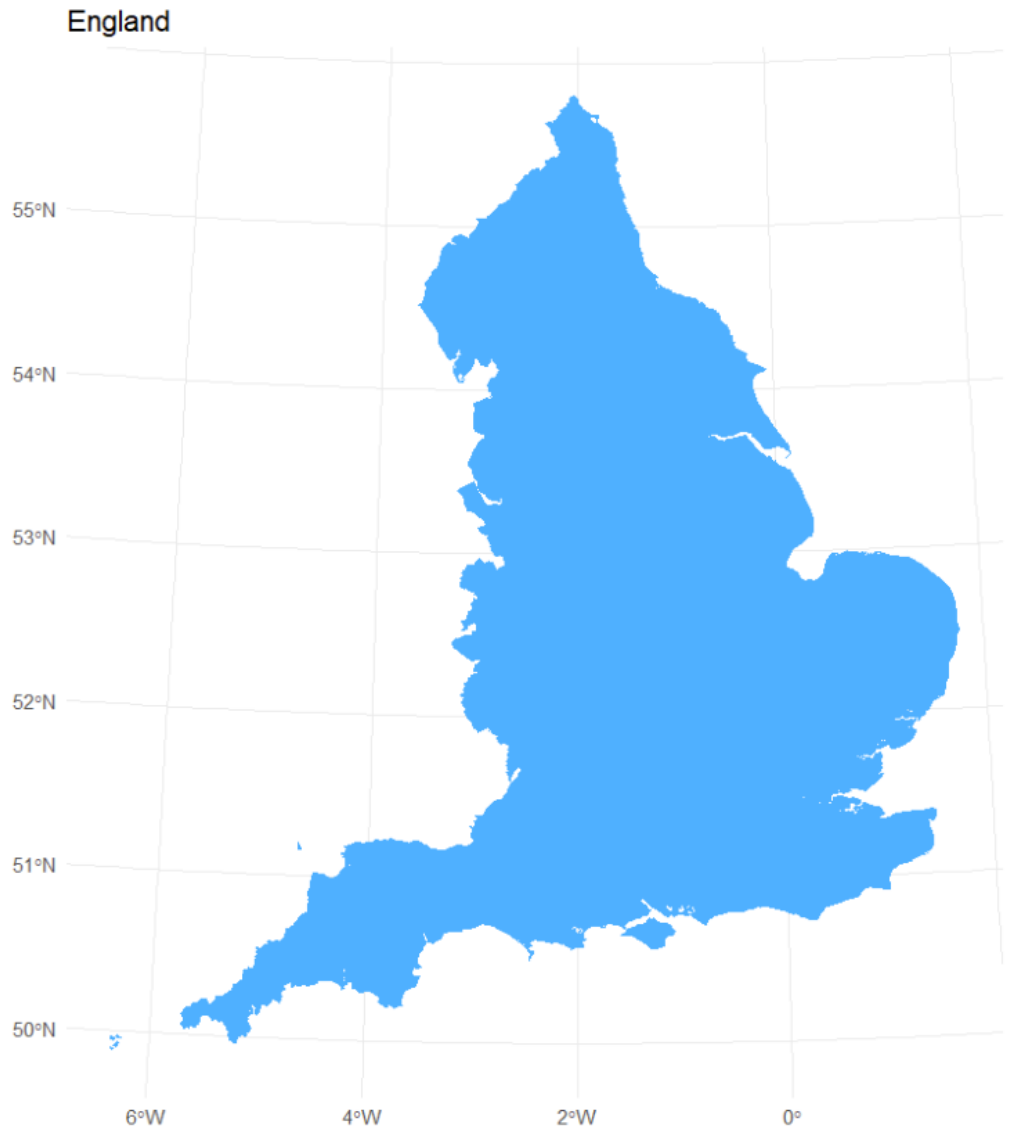


# A map of England

```
> cons_sf %>% filter(str_detect(PCON24CD, "^E"))->england_sf
>
> england_sf
Simple feature collection with 543 features and 8 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 87323.11 ymin: 7054.1 xmax: 655653.8 ymax: 657549.7
Projected CRS: OSGB36 / British National Grid
# A tibble: 543 × 9
   PCON24CD  PCON24NM                 PCON24NMW  BNG_E  BNG_N   LONG   LAT GlobalID                            geometry
 * <chr>     <chr>                    <chr>      <dbl>  <dbl>  <dbl> <dbl> <chr>                                 <MULTIPOLYGON [m]>
 1 E14001063 Aldershot                NA        484716 155270 -0.786  51.3 b6eea658-887c-43bd-81a6-e87… (((485406.9 159918.6, 48…
 2 E14001064 Aldridge-Brownhills      NA        404720 301030 -1.93   52.6 d1f29811-2c62-4dcb-b0dc-06d… (((406519.1 305054.3, 40…
 3 E14001065 Altrincham and Sale West NA        374132 389051 -2.39   53.4 5345726c-b6b4-4d12-a310-8b2… (((379104.1 393143.9, 37…
 4 E14001066 Amber Valley             NA        440478 349674 -1.40   53.0 4c232f33-f30b-4526-bfec-d5a… (((444868.4 353958.1, 44…
 5 E14001067 Arundel and South Downs  NA        497309 118530 -0.616  51.0 9eb29363-b47b-463a-9698-bc8… (((523813.2 118212, 5247…
 6 E14001068 Ashfield                 NA        450035 356564 -1.25   53.1 c6664362-7ca9-43cf-9920-222… (((455809 357962.2, 4551…
 7 E14001069 Ashford                  NA        611218 142572  1.02   51.1 d251384e-d248-457f-89da-a0b… (((620103.1 146702.2, 61…
 8 E14001070 Ashton-under-Lyne        NA        392654 398807 -2.11   53.5 aac16076-67e4-4ab4-aae2-a41… (((393565.7 396317.2, 39…
 9 E14001071 Aylesbury                NA        495172 216598 -0.620  51.8 975e6733-c9b0-4936-96b2-869… (((489373.9 224192.7, 48…
10 E14001072 Banbury                  NA        438918 232636 -1.43   52.0 36a7bbd3-7140-406a-854d-aa1… (((451341.2 245116.2, 45…
# i 533 more rows
# i Use `print(n = ...)` to see more rows
```

# A map of England

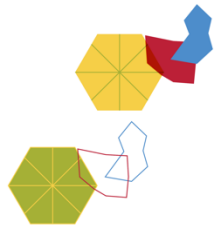
```
ggplot(england_sf) +
  geom_sf(fill = "#33afff", color = "#33afff")+
  theme_minimal()+
  labs(title = "England")
```



# Preparing the 14 zones

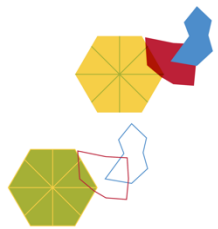
- For this step, we want to import the 14 zones, and also find their centroids so we can label them appropriately on our map

```
living_zones<-read_sf("Data/Biogeographic_Zones_Living_England_(2021)___Natural_England.shp")

living_zones$centroid <- st_centroid(living_zones$geometry)

centroids_living <- st_coordinates(living_zones$centroid)

living_zones <- cbind(living_zones, centroids_living)
```



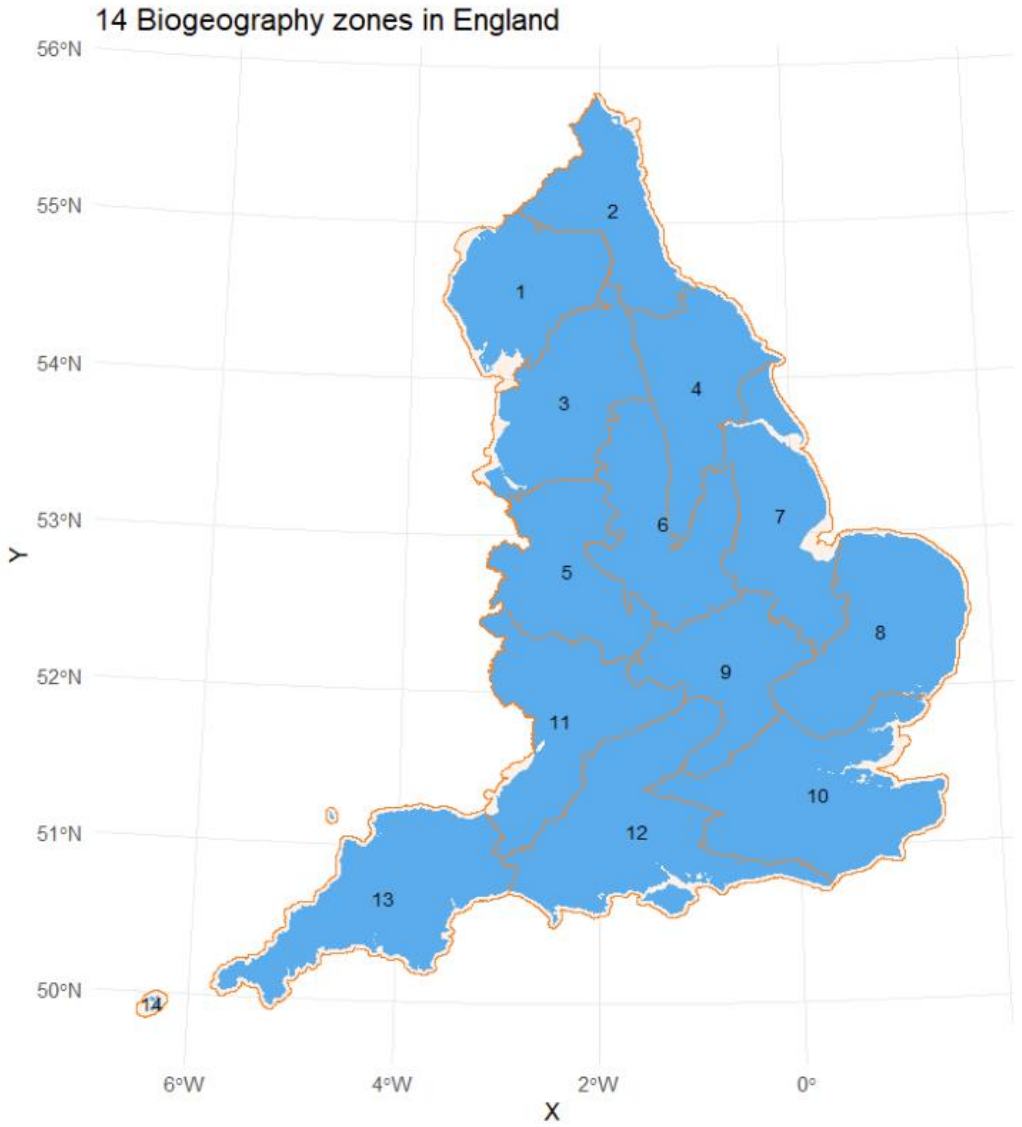
# Preparing the 14 zones

```
> living_zones
Simple feature collection with 14 features and 6 fields
Active geometry column: geometry
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 77134.07 ymin: 2099.38 xmax: 658653.8 ymax: 660601.5
Projected CRS: OSGB36 / British National Grid
First 10 features:
   Zone  Area Perimeter                              GlobalID        X        Y                       geometry
1     1  9010       528 014d9600-1716-43c4-a6fd-1503df3d3666 345880.9 520419.4 MULTIPOLYGON (((344099.1 57...
2     2  8070       581 3d8840c4-ce11-420d-b30b-c938917eadd7 409708.9 576996.3 MULTIPOLYGON (((399683.7 66...
3     3  9945       588 a9e176d4-5dd4-40d0-b6c3-82557f73c739 375967.8 440478.6 MULTIPOLYGON (((406364.3 52...
4     4  9024       582 acd264e8-b12b-459e-883f-2ea0fb49241d 467808.6 451417.1 MULTIPOLYGON (((468803.3 52...
5     5  9069       711 c09ba998-28d0-4374-8dda-59076abb20bf 378079.4 319643.7 MULTIPOLYGON (((330130.3 29...
6     6  9821       749 218eeeab-6f74-413c-921c-d73ad10bbdd0 444255.7 353423.7 MULTIPOLYGON (((437243.3 44...
7     7 10504       673 ae3b0a4c-ea07-4767-9dae-9316612781fd 526762.0 359629.9 MULTIPOLYGON (((518868.1 46...
8     8 12136       525 2724af34-7050-4f99-a02f-139fd172057f 596355.7 276748.7 MULTIPOLYGON (((627706.6 22...
9     9  7989       565 c5a04597-77e6-4c91-b28e-abcf5b8fd57b 488243.0 249312.3 MULTIPOLYGON (((549504.6 26...
10   10 15154       741 74200f43-ccd0-45e4-b366-46e648aee64d 552299.9 160919.4 MULTIPOLYGON (((615364.6 23...
                  centroid
1  POINT (345880.9 520419.4)
2  POINT (409708.9 576996.3)
3  POINT (375967.8 440478.6)
4  POINT (467808.6 451417.1)
5  POINT (378079.4 319643.7)
6  POINT (444255.7 353423.7)
7    POINT (526762 359629.9)
8  POINT (596355.7 276748.7)
9    POINT (488243 249312.3)
10 POINT (552299.9 160919.4)
```



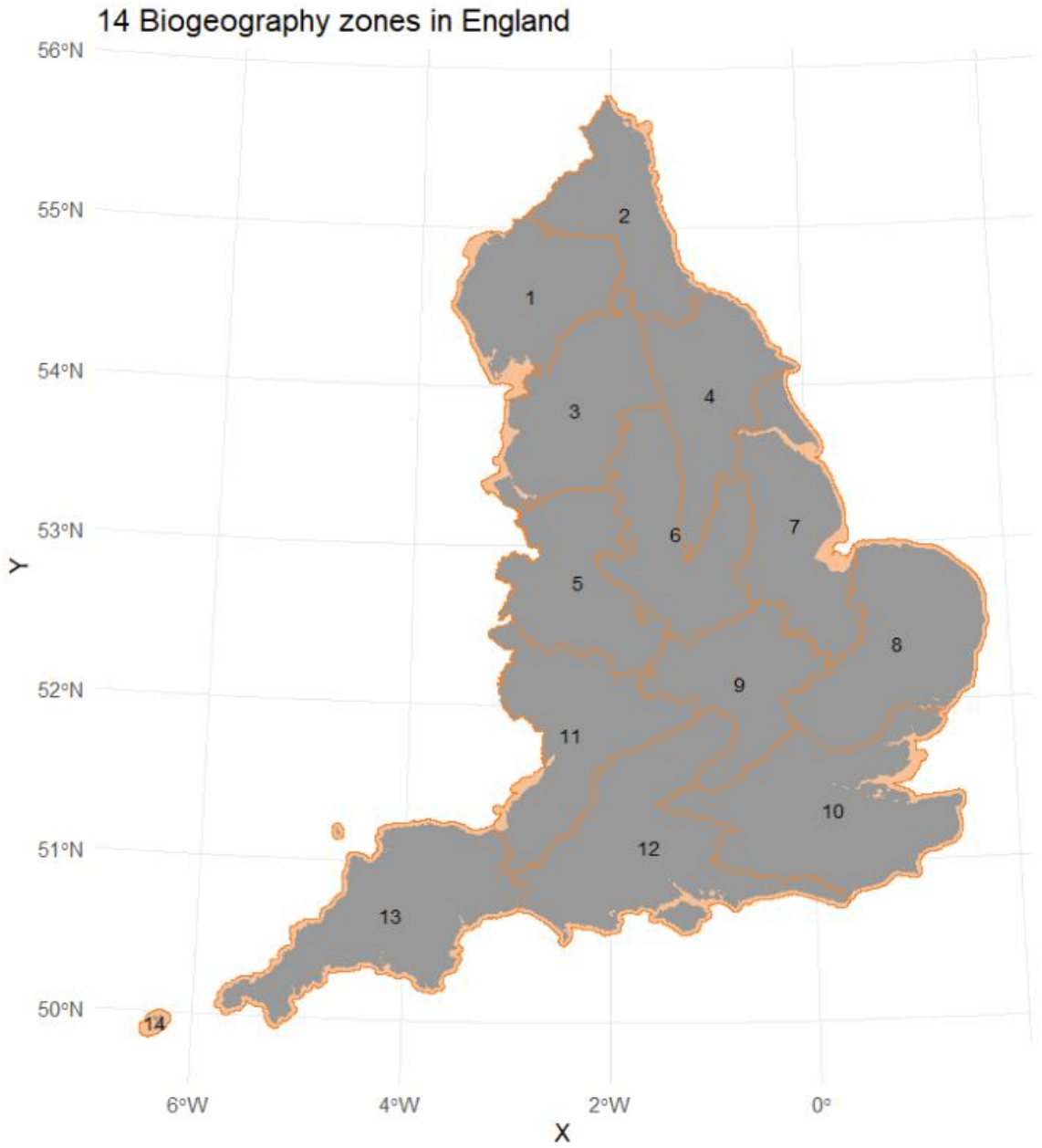
# Adding the new layer

```
ggplot(england_sf) +
  geom_sf(fill = "#33afff", color = "#33afff") +
  geom_sf(data=living_zones, color="#ff8333", fill="#ff8333",alpha=0.1)+
  geom_text(data=living_zones,aes(x = X, y = Y, label = Zone),
            size = 3, color = "black")+
  labs(title = "14 Biogeography zones in England")+
  theme_minimal()
```



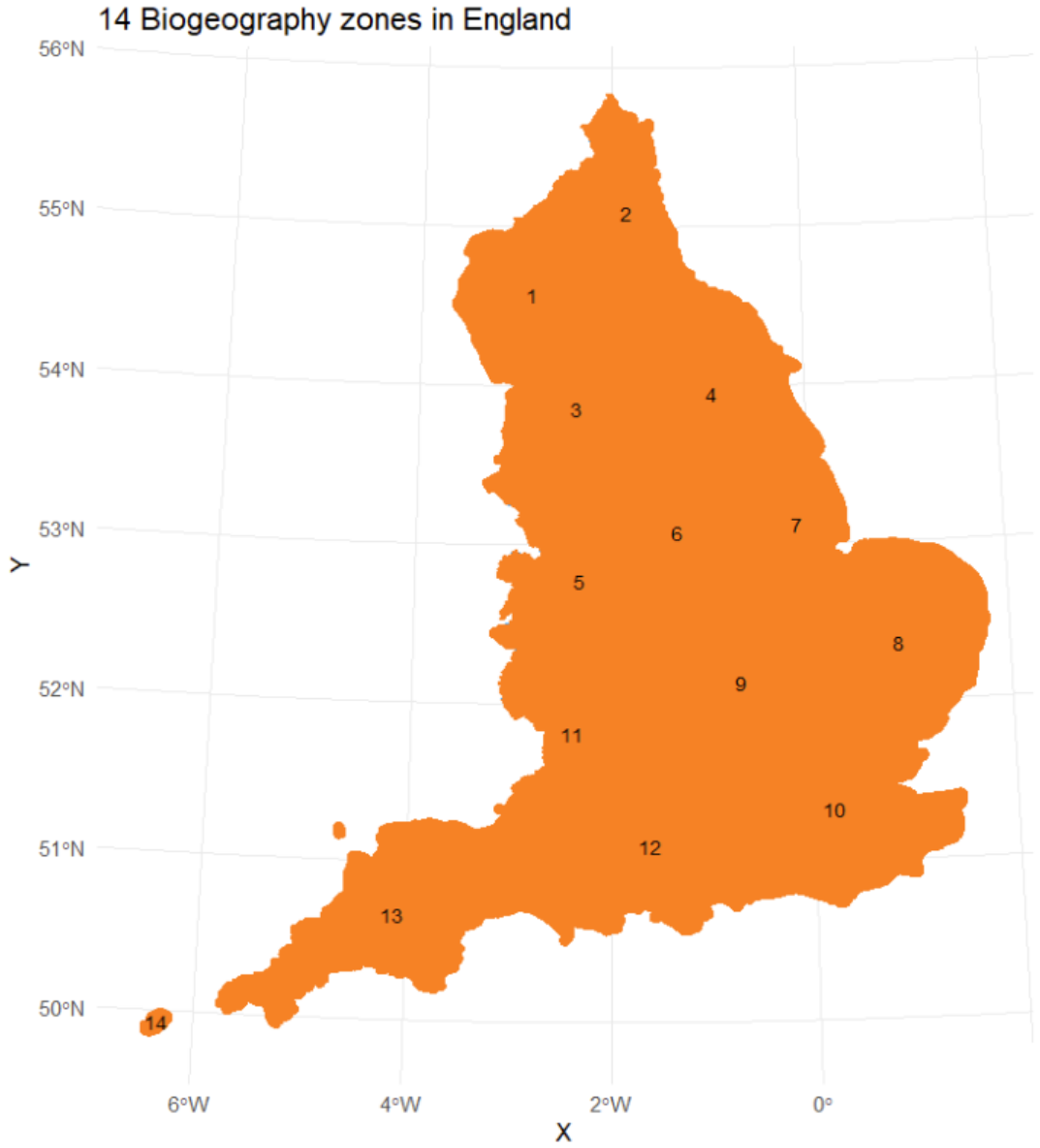
# Playing with alpha

```
ggplot(england_sf) +
  geom_sf(fill = "#33afff", color = "#33afff") +
  geom_sf(data=living_zones, color="#ff8333", fill="#ff8333",alpha=0.5)+
  geom_text(data=living_zones,aes(x = X, y = Y, label = Zone),
            size = 3, color = "black")+
  labs(title = "14 Biogeography zones in England")+
  theme_minimal()
```



# Playing with alpha

```
ggplot(england_sf) +
  geom_sf(fill = "#33afff", color = "#33afff") +
  geom_sf(data=living_zones, color="#ff8333", fill="#ff8333",alpha=1)+
  geom_text(data=living_zones,aes(x = X, y = Y, label = Zone),
            size = 3, color = "black")+
  labs(title = "14 Biogeography zones in England")+
  theme_minimal()
```



# Adding some context

- Let us add Scotland and Wales back on our illustration to make the zones stand out more

```
cons_sf %>%
  filter(str_detect(PCON24CD, "^E")|
           str_detect(PCON24CD, "^S")|
           str_detect(PCON24CD, "^W"))->engScotWal_sf

ggplot(engScotWal_sf) +
  geom_sf(fill = "#33afff", color = "#33afff") +
  geom_sf(data=living_zones, color="#ff8333", fill="#ff8333",alpha=0.1)+
  geom_text(data=living_zones,aes(x = X, y = Y, label = Zone),
            size = 3, color = "black")+
  labs(title = "14 Biogeography zones in England")+
  theme_minimal()
```