Visualising data with R

Practical exercises.

In [2]:
```
# Load in some packages for plotting and data manipulation.
library(ggplot2)
library(ggthemes)
library(data.table)
```

0. Data manipulation

A note about datatypes. R has a built-in type for data frames, called `data.frame` . There is a package which provides a more fully-featured data frame, called `data.table` . I recommend using the latter, and it's used throughout this practical.

- When you load a dataset with the `fread` command, it creates a `data.table`
- When you pass a pandas `DataFrame` to R using rpy2, it creates a `data.frame` . Use `df <- data.table(df)` to convert it into a `data.table` .
- ggplot2 works with both. But some of the data manipulation commands in this practical only work with `data.table` .

You can't get away with no R at all. Here is a small session of R data manipulation.

In [ ]:
```
# Create a vector of integers, including a missing value
x <- c(3,5,NA,2)

# Read in a dataset in CSV format
iris <- fread('https://teachingfiles.blob.core.windows.net/founds/iris.csv')

# How many rows does it have?
nrow(iris)

# Print out some sample rows
iris[sample(nrow(iris),4)]

# Select some rows, using a boolean vector, then subselect with an integer vector
iris[Sepal.Length>=6 & Species=='virginica'][1:5]

# Compute something based on a column. In R, '.' is just another character with no special meaning.
mean(iris[, Sepal.Length])

# Modify the dataframe by defining a new column
iris[, Sepal.Area := Sepal.Length * Sepal.Width]

# Create a new dataframe, with a subset of rows and columns
df2 <- iris[Sepal.Area >= quantile(Sepal.Area,.9), list(Species, Petal.Length, Petal.Width, Sepal.Are
```

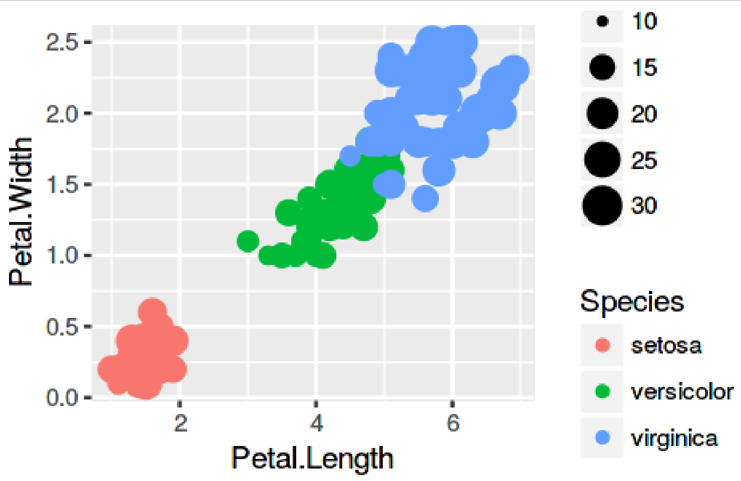
1. Scatterplot point size

The Iris dataset was collected by the botanist Edgar Anderson and popularized by Ronald Fisher in 1936. Fisher has been described as a "genius who almost single-handedly created the foundations for modern statistical science". The dataset consists of 50 samples from each of three species of iris, each with four measurements.

In [11]:
```
# Fetch a dataframe in CSV format, and print out a sample of rows
iris <- fread('https://teachingfiles.blob.core.windows.net/founds/iris.csv')
iris[sample(nrow(iris),3)]
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
6.4	3.1	5.5	1.8	virginica
6.9	3.1	4.9	1.5	versicolor
5.7	2.6	3.5	1.0	versicolor

In [13]:
```
# A simple scatterplot
options(repr.plot.width=5, repr.plot.height=2.5)
ggplot(data=iris) +
    geom_point(aes(x=Petal.Length, y=Petal.Width, size=Sepal.Length*Sepal.Width, col=Species))
```



Exercise. I want to make the points smaller, to reduce overlap. Try `size=Sepal.Length*Sepal.Width*0.5` . Explain why it doesn't actually make the sizes smaller. ■

Exercise. Another problem with the size scale is that it's misleading: size 30 is not three times larger than size 10. Explain why this is. Look up the reference for `scale_size()` (http://ggplot2.tidyverse.org/reference/scale_size.html) and fix it. ■

Try this 'frogspawn' plot. The low-opacity area show smoothed density, and the high-opacity points show fine-grained density. Also, note another ggplot trick: you can define aesthetic mappings in the main `ggplot()` command, and they become the defaults for all the geoms.

```
ggplot(data=iris, aes(x=Petal.Length, y=Petal.Width, col=Species)) +
    geom_point(aes(size=Sepal.Length*Sepal.Width), alpha=.2, stroke=0) +
    geom_point(size=.1)
```

2. Positioning discrete scales

This climate dataset consists of monthly readings from a variety of weather stations around the UK. It is provided by the Met Office (https://www.metoffice.gov.uk/public/weather/climate-historic/#?tab=climateHistoric). The fields are

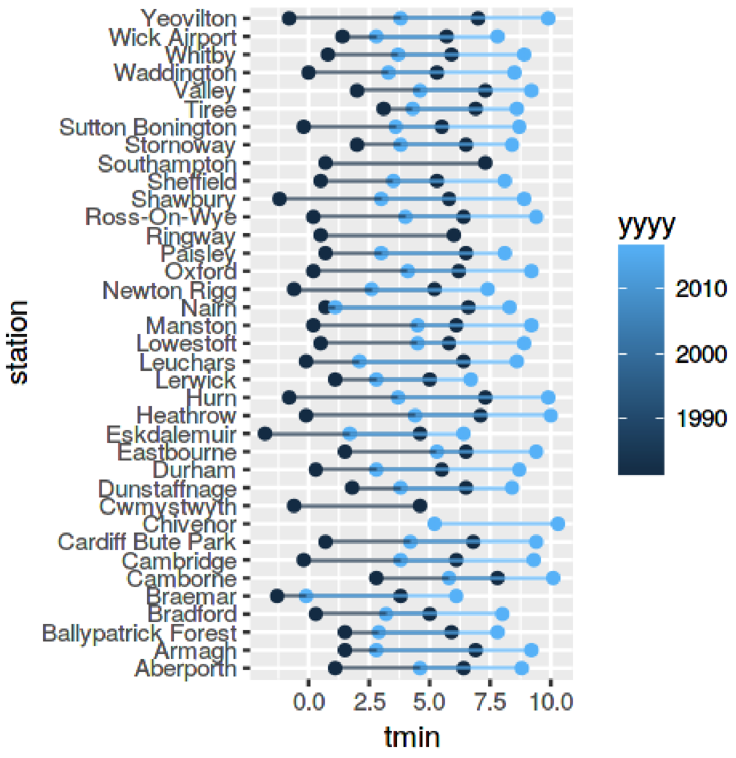
- yyyy, mm: year and month
- tmin, tmax, af, rain, sun: average temperature range in degrees C, days of air frost, total rainfall (units?), total sunshine duration (units?)
- station: the name of the monitoring station
-
    - lat, lng, height (in meters), country: attributes of the station

In [3]:
```
climate <- fread('https://teachingfiles.blob.core.windows.net/datasets/climate.csv')
climate[sample(nrow(climate),3)]
```

yyyy	mm	tmax	tmin	af	rain	sun	status	station	lat	lng	height	country
1932	2	7.7	1.0	11	3.3	73.7		Armagh	54.352	-6.649	62	N.Ireland
1966	8	NA	NA	---	106.7	199.8		Chivenor	51.089	-4.147	6	England
1968	2	4.4	-0.4	17	42.8	60.5		Sheffield	53.381	-1.490	131	England

Here is a dumbbell plot, showing the min-max range in 1985 and in 2017. The function `is.na(x)` applies to a vector `x` , and returns a boolean vector of the same length indicating whether each element is missing.

In [4]:
```
# Take readings in Feb 1981 and 2017, and discard rows with missing tmin or tmax
climate2 <- climate[yyyy %in% c(1981,2017) & mm==2 & !is.na(tmin) & !is.na(tmax)]

options(repr.plot.width=4, repr.plot.height=4)

ggplot(data=climate2, aes(col=yyyy)) +
    geom_point(aes(x=tmin, y=station)) +
    geom_point(aes(x=tmax, y=station)) +
    geom_segment(aes(x=tmin, xend=tmax, y=station, yend=station), alpha=.7)
```



Exercise. ggplot has interpreted the colour scale as numeric (since 1981 and 2017 are numbers), but it makes more sense here to treat them as categorical. Replace `aes(col=yyyy)` with `aes(col=as.character(yyyy))` and see what happens. Compare to `aes(col=factor(yyyy))` . ▰

The `as.character(x)` function converts a vector of any type into a vector of strings (which R calls 'character vector'). The `factor(x)` function converts a vector of any type into a vector of enum values, and it picks up the levels of the enum from the values present in `x` . You can run `summary(df)` to report the type of each column in a dataframe. If you're using rpy2, you'll often end up with string vectors, since Python has poor enum support.

Exercise. There are several things wrong with this plot.

- The labels for the x-axis and the colour guide aren't helpful. To fix them, see `labs()` (http://ggplot2.tidyverse.org/reference/labs.html).
- The line segments are on top of the points, which is ugly. Geoms are plotted in the order you specify them. Rearrange the geoms to put the points last. ▰

There are two more things wrong, which will take a bit more work to fix. First, the vertical axis is a waste. We should use it to convey information, e.g. by sorting the stations non-alphabetically. Second, the bars overlap, which looks bad and is unhelpful. A better plot would either jitter the bars, or use colour-coding differently.

We'll rearrange the stations on the y-axis. To do this, we need to ensure that `station` is a vector of enums, and we need to put the enum levels in the order we want. If you're using rpy2 you can create an order as a list of strings, and pass it into R. Here, we'll use R instead. Run the following code to produce a character vector (i.e. list of strings) called `station_order` .

In [5]:
```
df <- dcast(climate2[, list(yyyy,tmin,tmax,station)], station ~ yyyy, value.var=c('tmin','tmax'))
df[, d := pmax(tmax_2017-tmax_1981, tmin_2017-tmin_1981)]
station_order = df[order(d), station]
print(station_order)
```

```
 [1] "Tiree"              "Lerwick"            "Nairn"
 [4] "Ballypatrick Forest" "Stornoway"          "Dunstaffnage"
 [7] "Wick Airport"       "Leuchars"           "Armagh"
[10] "Braemar"            "Paisley"            "Valley"
[13] "Bradford"           "Camborne"           "Sheffield"
[16] "Whitby"             "Durham"             "Newton Rigg"
[19] "Waddington"         "Aberporth"          "Cardiff Bute Park"
[22] "Eskdalemuir"        "Eastbourne"         "Ross-On-Wye"
[25] "Sutton Bonington"   "Oxford"             "Cambridge"
[28] "Lowestoft"          "Shawbury"           "Manston"
[31] "Heathrow"           "Hurn"               "Yeovilton"
[34] "Chivenor"           "Cwmystwyth"         "Ringway"
[37] "Southampton"
```

Now, define a new column in the dataset called `stationO`, which will have exactly the same values as `station` --- but they will be marked as being of type enum, with a specific ordering of the levels.

In [6]:
```
climate2[, stationO := factor(station, levels=station_order)]
```

Exercise. Replace `station` with `stationO` in your ggplot, and check that the y-axis has been reordered. ▰

Exercise. Try producing separate panels by country (England, Wales, N.Ireland, Scotland). Try adding

```
+ facet_wrap(~country, nrow=1)
```

Try instead

```
+ facet_grid(country~., scales='free_y', space='free_y')
```

Try these commands with and without the optional arguments (i.e. nrow, scales, space). Look up the help for `facet_wrap()` (http://ggplot2.tidyverse.org/reference/facet_wrap.html) and `facet_grid()` (http://ggplot2.tidyverse.org/reference/facet_grid.html) and work out why they do what they do. ▰

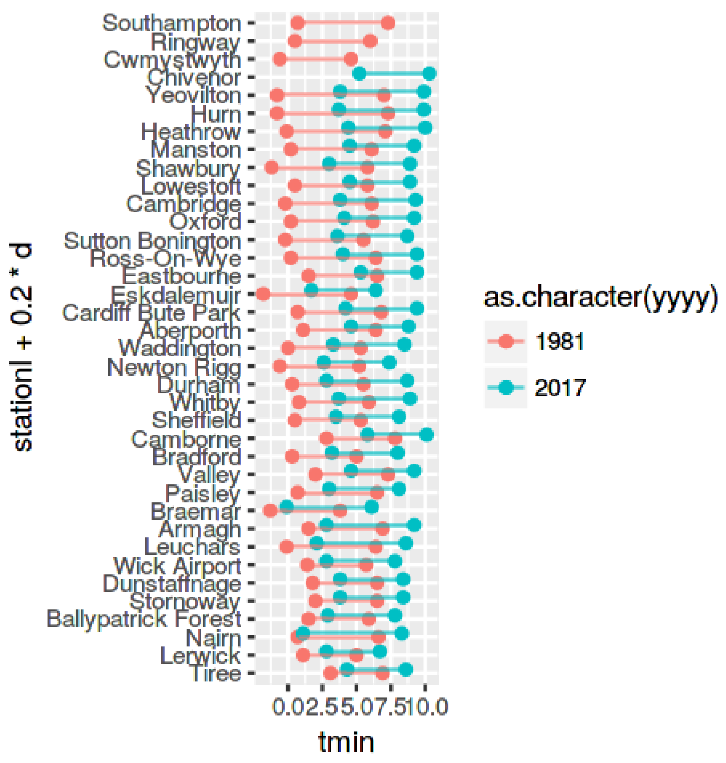
Now we'll jitter the two bars, 1981 vs 2017.

There are commands for adjusting the position (http://ggplot2.tidyverse.org/reference/#section-layer-position-adjustment) of geoms, such as `position_dodge()` and `position_jitter()`. With these position modifiers, ggplot first computes the nominal position of the geom, then it modifies positions in the display. In other words, position modifiers act in coordinate space rather than affecting the aes scale. However, this mechanism only works for one geom at a time. Here we have three geoms that make up our shape, and we want the points and the lines to have exactly the same position modification. ggplot doesn't support this (unless you create your own new composite geom, which isn't pleasant).

The simplest solution is to manage the scale ourselves.

- First, let `stationI` be an integer for each station. (Earlier we used `stationO := factor(station)` to convert `station` from a character vector (i.e. strings) to a factor (i.e. enums). Now we use `as.numeric(stationO)` converts enum levels into integer codes.)
- Next, define a column `d` containing the amount by which we wish to jitter.
- Plot the points and lines, using manually jittered y-coordinates.
- Add labels on the y-axis, one for each station label. See the help for `scale_y_continuous()` (http://ggplot2.tidyverse.org/reference/scale_continuous.html) for how to customize ticks.

In [10]:
```
climate2[, stationI := as.numeric(stationO)]
climate2[, d := ifelse(yyyy==2017, 1, 0)]
unique_stations = levels(climate2[, stationO])

ggplot(data=climate2, aes(col=as.character(yyyy))) +
    geom_point(aes(x=tmin, y=stationI + 0.2*d)) +
    geom_point(aes(x=tmax, y=stationI + 0.2*d)) +
    geom_segment(aes(x=tmin, xend=tmax, y=stationI + 0.2*d, yend=stationI+0.2*d), alpha=.7) +
    scale_y_continuous(breaks=1:length(unique_stations), labels=unique_stations, minor_breaks=NULL, e
```



Exercise. Apply jittering to the `facet_grid` version of this plot, which has one panel per country. ▰

Exercise. Instead of jittering the bars, display using a three-fold colour scheme. Use green for "this temperature range was seen in 1981 only", use grey for "this temperature range was seen in both 1981 and 2017", and use purple for "this temperature range seen in 2017 only". Use the blending trick (from the introductory lecture) to get entries in the legend for each of the colours. Use `scale_colour_manual()` (http://ggplot2.tidyverse.org/reference/scale_manual.html) to create a discrete colour scale with custom colours. ▰

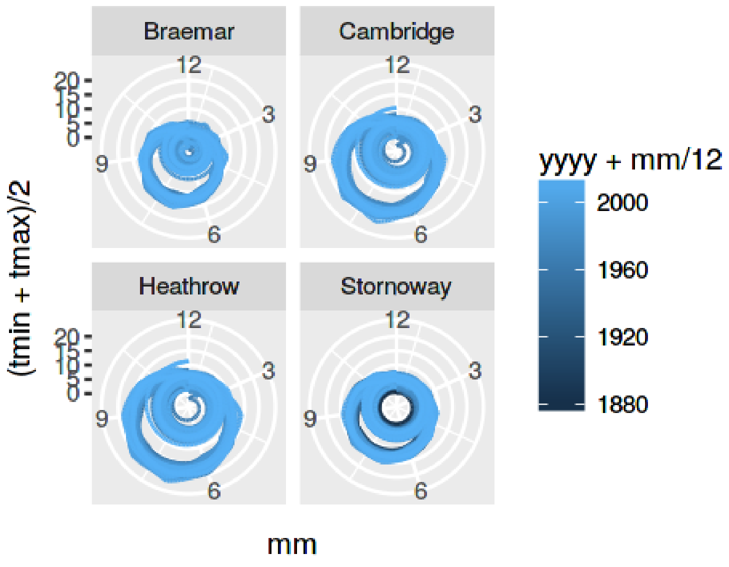
3. Coordinate systems

We'll use the same climate data as in Section 2.

In [ ]:
```
climate <- fread('https://teachingfiles.blob.core.windows.net/datasets/climate.csv')
climate[sample(nrow(climate),3)]
```

Temperatures seem to be steadily rising, but they also follow an annual cycle. A neat plot for showing this combination of 'secular trend' and 'periodic pattern' is a spiral plot: polar coordinates, angle to show the month, radius to show the temperature. We expect to see a spiral pattern, showing increasing temperatures for every month. Here's a first attempt, using `geom_path`, which draws a line connecting all the points you supply.

In [12]:
```
df <- climate[station %in% c('Cambridge', 'Heathrow', 'Braemar', 'Stornoway')]

options(repr.plot.width=4, repr.plot.height=3)

ggplot(data=df) +
    geom_path(aes(x=mm, y=(tmin+tmax)/2, col=yyyy+mm/12)) +
    facet_wrap(~station) +
    coord_polar(theta='x')
```



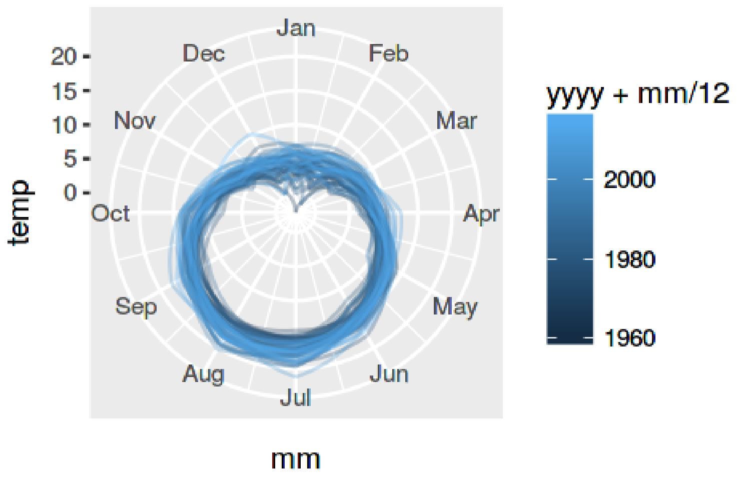
Exercise.

- The x (theta) axis looks silly. Set the ticks using `scale_x_continuous()` (http://ggplot2.tidyverse.org/reference/scale_continuous.html). You'll need a character vector (i.e. list of strings) for the months: `months <- c('Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec')`
- The aes scale has put mm=1 at 0 degrees and mm=12 at 360 degrees. Explain why it did this. Modify the scale so that the x-range goes from mm=0 to mm=12.
- Something's not right! Plot just one station, over a span of 2.5 years, to see more clearly how the `geom_path` is connecting points. ▰

The problem is that we told it to draw a geom_path, and this geom draws a straight line between successive points. The straight line from x=mm=12 (Dec 2015) to x=mm=1 (Jan 2016) goes via x=11,10,...,2,1. (You can see this more clearly on a conventional plot with Cartesian coordinates.)

To remedy the problem, we need to give up on geom_path, and its perfectly logical but unhelpful notion of 'straight line interpolation'. Instead, let's create each line segment explicitly, e.g. a line segment from (x=12, y=temp[Dec 2015]) to (x=13, y=temp[Jan 2016]). We need to create our own dataframe for this, containing for each row the temperature this month and the temperature the following month.

In [14]:
```
# Advanced R data manipulation code. You can do it in Python if you prefer!
df <- copy(climate)
df[, temp := (tmin+tmax)/2]
df <- df[order(yyyy,mm)]  # sort the data frame by (yyyy,mm)
df[, temp_next := c(temp[2:.N], NA), by=station]  # within each station, let temp_next be lagged vers
df <- df[!is.na(temp) & !is.na(temp_next)]

ggplot(data=df[station=='Cambridge']) +
    geom_segment(aes(x=mm, xend=mm+1, y=temp, yend=temp_next, col=yyyy+mm/12), alpha=.3) +
    coord_polar(theta='x') +
    scale_x_continuous(breaks=1:12, labels=months, limits=c(1,13))
```



Exercise. This plot isn't showing us clearly enough the overall temperature trend. Instead, create several panels, one per decade. Within each panel, show the spiral for the entire dataset in light grey, and show that decade's values in blue. Hopefully, this will let us situate the temperatures each decade within the context of the entire dataset.

4. Maps, overplotting, and zooming

Here are two datasets from Game of Thrones. They are both denormalized. The first, `action` , contains details of every scene. It was derived from data assembled by Jeffrey Lancaster (https://medium.com/@jeffrey.lancaster/the-ultimate-game-of-thrones-dataset-a100c0cf35fb). The columns are

- season, episode, scene: the scene counts up within each (season,episode)
- locseq: the location_id for that scene, that goes roughly north-to-south and keeps related locations together
  - loc, subloc: the name of the location
  - ID: a key to the map dataset
  - lat, long: coordinates
- t, tend: start end end time of the scene, in seconds since the start of the episode
- character, title, house, alive: one row for each character appearing in this scene

The second, `gotmap` , contains geo data. It was derived from a shapefile created by cadaei of the cartographers' guild (https://www.cartographersguild.com/showthread.php?t=30472) and licensed under the CC BY-NC-SA 3.0 (http://creativecommons.org/licenses/by-nc-sa/3.0/). The columns are

- layer, geom: the geo data consists of several layers, each with a shape type (geom=1: POINT, geom=3: MULTILINE, geom=5: POLYGONS)
- ID, id, name, type: each shape within the layer consists of multiple shapes, each with a globally unique ID
  - piece, group: each POLYGONS or MULTILINE consists of multiple pieces; `piece` is the piece number, and `group=(ID,piece)`
  - order, lat, long: the points within each shape

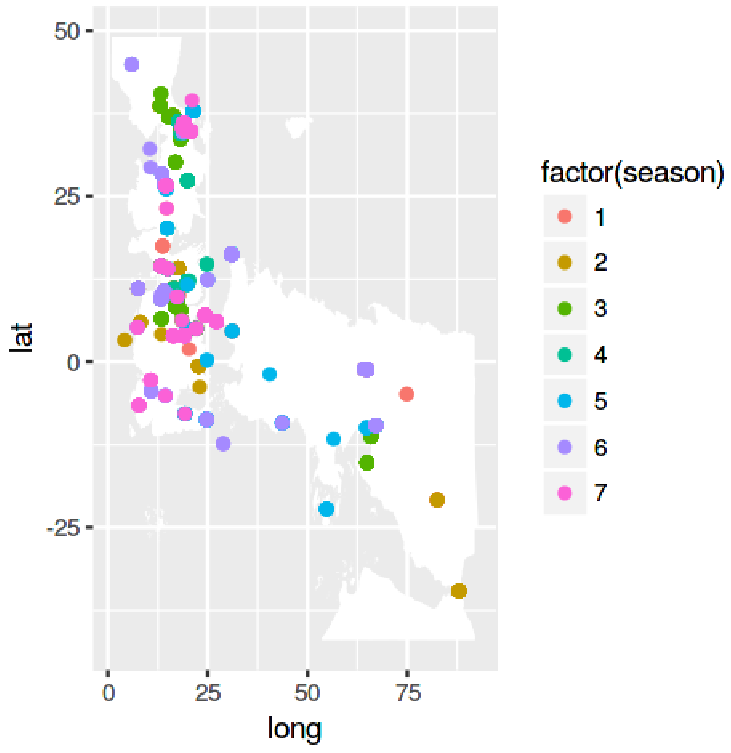
In [15]:
```
action <- fread('https://teachingfiles.blob.core.windows.net/datasets/got_activity.csv')
gotmap <- fread('https://teachingfiles.blob.core.windows.net/datasets/got_map.csv')
action[sample(nrow(action),3)]
gotmap[sample(nrow(gotmap),3)]
```

season	episode	scene	loc	subloc	locseq	ID	lat	long	t	tend	character	title	house	alive
6	10	63	The Reach	Oldtown	91	355	-6.57124	7.752675	1922	2048	Citadel Maester			TRUE
1	10	7	The Riverlands	Camp of the North	62	NA	10.65361	14.078230	367	487	Robb Stark		Stark	TRUE
3	10	37	The Wall	Castle Black	19	207	34.89038	18.853961	3098	3139	Jon Snow		Stark	TRUE

layer	geom	ID	name	type	piece	group	order	lat	long
river	3	431	NA	NA	3	431.3	28	13.623858	19.53586
continent	5	2	Essos	NA	1	2.1	4224	8.278365	65.32570
continent	5	1	Westeros	NA	1	1.1	635	15.641844	22.78012

Here's a basic map of where scenes take place. For the map areas, we need to provide a `group` aes, to specify which points consitute a single polygon. The `group` column here is made up of shape ID and piece pasted together, so that if there's a single POLYGONS shape made up of multiple subpieces then ggplot knows not to draw them all connected together. This command plots all the map features with geom=5, i.e. of type POLYGONS.

In [20]:
```
# Get one row per scene
scenes <- unique(action[,list(scene,season,episode,loc,subloc,lat,long,t,tend)])

options(repr.plot.width=4, repr.plot.height=4)

ggplot(data=scenes, aes(x=long,y=lat)) +
   geom_polygon(data=gotmap[geom==5], aes(group=group), fill='white') +
   geom_point(aes(col=factor(season)))
```



(For real-world map plotting, you should use the ggmap package. Here's a [helpful tutorial (https://medium.com/fastah-project/a-quick-start-to-maps-in-r-b9f221f44ff3)](https://medium.com/fastah-project/a-quick-start-to-maps-in-r-b9f221f44ff3). This package makes it easy to download raster tiles from Google Maps or other sources, and to use them as a base under your own geoms. For Westeros, we have to draw our own maps from pure shapefiles.)

Exercise. Customize the scatterplot. First, set the size of each point to reflect the duration of the scene. Second, seasons are ordered, so use a Brewer colour scale. (This is a collection of well-chosen discrete colour scales, either sequential or diverging or qualitative.)

```
+ scale_colour_brewer(type='seq', palette='RdPu')
```

Third, many scenes happen at the same place, so set `alpha=0.1` for every point. Fourth, we don't want the legend to display at low opacity, so customize how the guide is shown:

```
+ guides(colour = guide_legend(override.aes=list(alpha=1)))
```

Read the help for guides: axes, and legends (http://ggplot2.tidyverse.org/reference/#section-guides-axes-and-legends) for more information. ◼

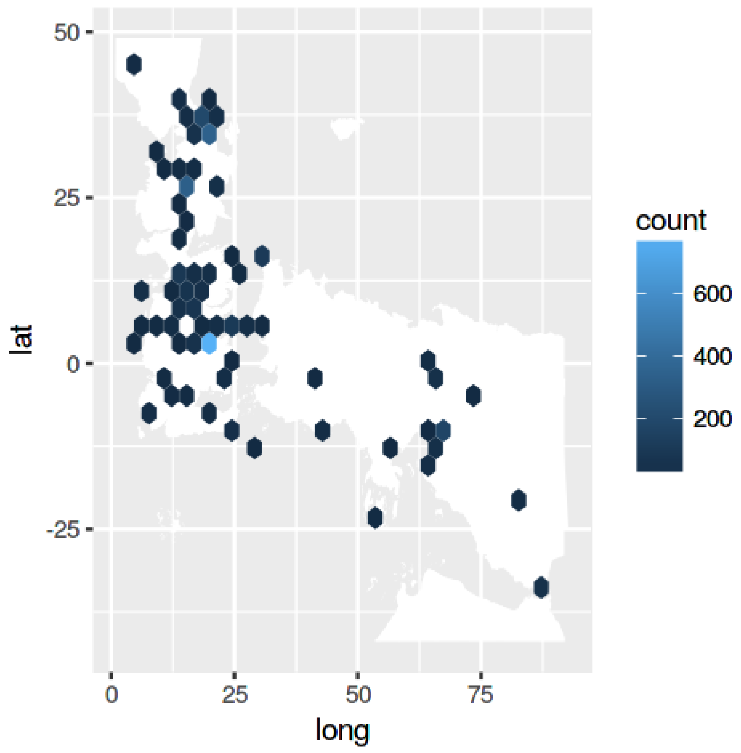
This plot, with a low alpha for each point, makes us realise that the overplotting problem is severe. It's almost impossible to get a sense of how much data there is. A common trick is to add jitter to each point, using the `position_jitter()` (http://ggplot2.tidyverse.org/reference/#section-layer-position-adjustment) modifier.

```
ggplot(data=scenes, aes(x=long,y=lat)) +
   geom_polygon(data=gotmap[geom==5], aes(group=group), fill='white') +
   geom_point(aes(col=factor(season)), position=position_jitter(width=1,height=1))
```

Exercise. Find a combination of point size and alpha and jitter amount that shows the data more clearly. Also, instead of `data=scenes` , use `data=scenes[sample(nrow(scenes))]` . This shuffles the rows of the dataframe, so that the points get drawn in random order, so we don't end up with all of season 7 overplotting all of season 1. ◼

Another solution to overplotting is to divide the area into grid cells, and display counts. A rectangular grid creates perceptual artefacts -- "it is known" -- and so a hexagonal grid is better. It's easy to create one, with the help of an auxiliary R package. (You probably need to install it first, with `install.packages('hexbin')` .) The following plot uses a `geom_hex()` (http://ggplot2.tidyverse.org/reference/geom_hex.html) geom, which comes with a default stat of `stat_bin_hex` . This is very similar to `geom_bar()` (http://ggplot2.tidyverse.org/reference/geom_bar.html), which comes with a default stat of `stat_count` .

In [21]:
```
library(hexbin)

ggplot(data=scenes, aes(x=long,y=lat)) +
   geom_polygon(data=gotmap[geom==5], aes(group=group), fill='white') +
   geom_hex()
```



Exercise. Replace the default `stat_bin_hex` with `stat_summary_hex` . This lets you compute an arbitrary function for each hex. Try

```
+ geom_hex(aes(z=tend-t), stat='summary_hex', fun='sum')
```

This uses the `stat_summary_hex` stat. It produces a new data frame, one row per hex, with an extra column called `..value..` , computed by applying `fun` to the vector of all `aes(z)` values of points in that hex. ▰

Exercise. This colour scale is completely dominated by King's Landing, which gets most screen time.

- Change the colour scale so it reports total screen time in hours.
- Use `aes(fill=pmin(..value.., 7))` to truncate screentime at 7 hours. (The function `pmin(x,y)` returns a pointwise minimum, similar to `numpy.minimum` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.minimum.html).) ▰

A better way to display an unbalanced set of colours would be to use a quantile scale. For example, let there be 5 colours on a Brewer sequential scale, let the lowest 20% of hexes get the bottom colour, the next 20% get the next colour, and so on. The best way to do this is to scale the values yourself, as we did in the climate example to jitter the bars. Here is a tutorial on custom hexbin (http://unconj.ca/blog/custom-hexbin-functions-with-ggplot.html), from a blogger with many helpful tips on R visualisation. (Or you could dive very deep into ggplot and define your own `scale_colour_quantile()` function.)

Finally, a note about zooming. There are two different ways to control the x-axis range.

- `scale_x_continuous(limits=c(xmin,xmax))` is applied at the aes mapping stage. Any value outside the limits is outside the scale, so it is discarded. If you have a polygon with one point outside this region, then the entire polygon will be discarded or otherwise messed up.
- `coord_cartesian(xlim=c(xmin,xmax))` is applied at the display stage. It specifies how the coordinate space is to be shown on the display. Any parts of a polygon lying outside the display will be clipped.

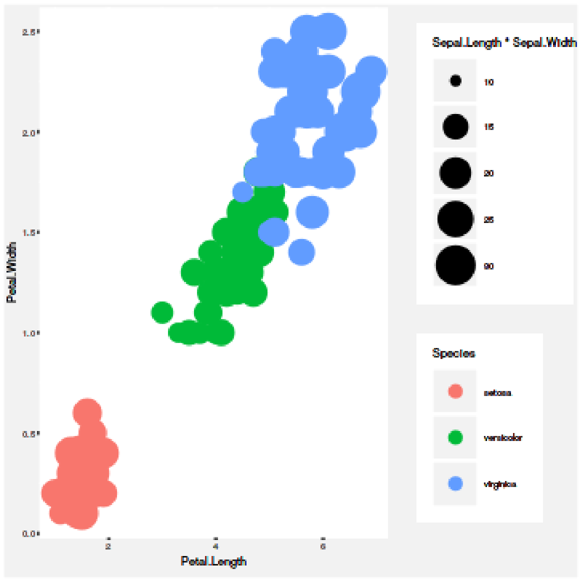
Exercise.

- Try both `scale_*_continuous` and `coord_cartesian` to show only latitudes -25 to 40, and longitudes 5 to 70.
- Also specify `geom_hex(binwidth=c(2,2))` so that the hexes are as wide as they are tall.
- Assuming that your plot size is still `options(repr.plot.width=4, repr.plot.height=4)` , you should see hexes that are tall and narrow. Explain why this is so, despite your setting `binwidth` .
- Look up other coordinate systems, and figure out how to make the hexes look regular. ▰

5. Finishing a plot

Themes (http://ggplot2.tidyverse.org/reference/#section-themes) control the display of all non-data elements of the plot. You can set many parameters in one go using a built-in theme like `theme_grey()` (http://ggplot2.tidyverse.org/reference/ggtheme.html). Or you can tweak individual settings using `theme()` (http://ggplot2.tidyverse.org/reference/theme.html). You can also combine these approaches.

In [48]:
```
options(repr.plot.width=3, repr.plot.height=3)  # specify plot size, in inches

ggplot(data=iris) +
    geom_point(aes(x=Petal.Length, y=Petal.Width, size=Sepal.Length*Sepal.Width, col=Species)) +
    theme_gray(base_size=4) +  # Set the base font size in points; there are 72.27 points per inch
    theme(plot.background = element_rect(fill='grey95'),
          panel.background = element_rect(fill='white'))
```



Here is how to save a plot.

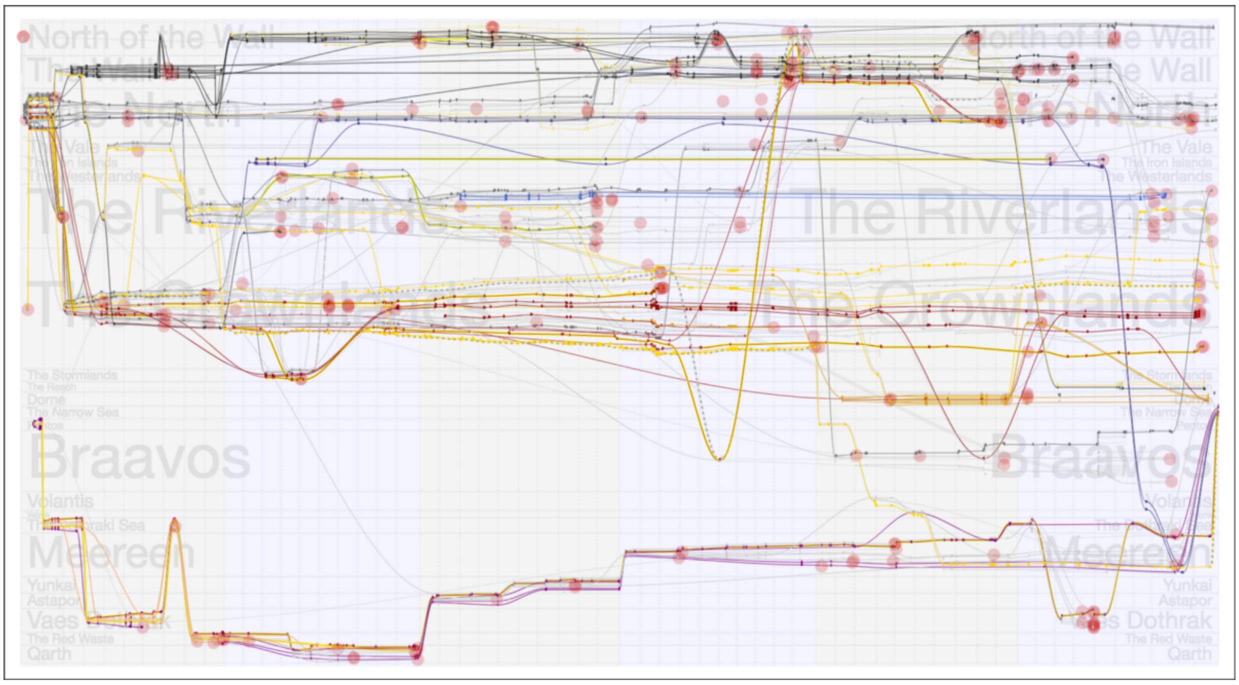
```
g <- ggplot(data=iris) +
    geom_point(aes(x=Petal.Length, y=Petal.Width, size=Sepal.Length*Sepal.Width, col=Specie
s)) +
    theme_gray(base_size=4) +  # Set the base font size in points; there are 72.27 points pe
r inch
    theme(plot.background = element_rect(fill='grey90', colour=NA),
          panel.background = element_rect(fill='grey65', colour=NA))

ggsave(filename='myplot.pdf', g)
ggsave(filename='myplot.png', g, width=4, height=4, units='in', dpi=300)
```

Exercise. Using `element_rect(fill='transparent')` , save the map of Westeros as a PNG with a transparent background. In addition to setting `theme(plot.background)` , you need to save it with `ggsave(..., bg='transparent')` . (Here, `bg='transparent'` is a core R graphics option, used by the png device driver, not part of ggplot.) ▰

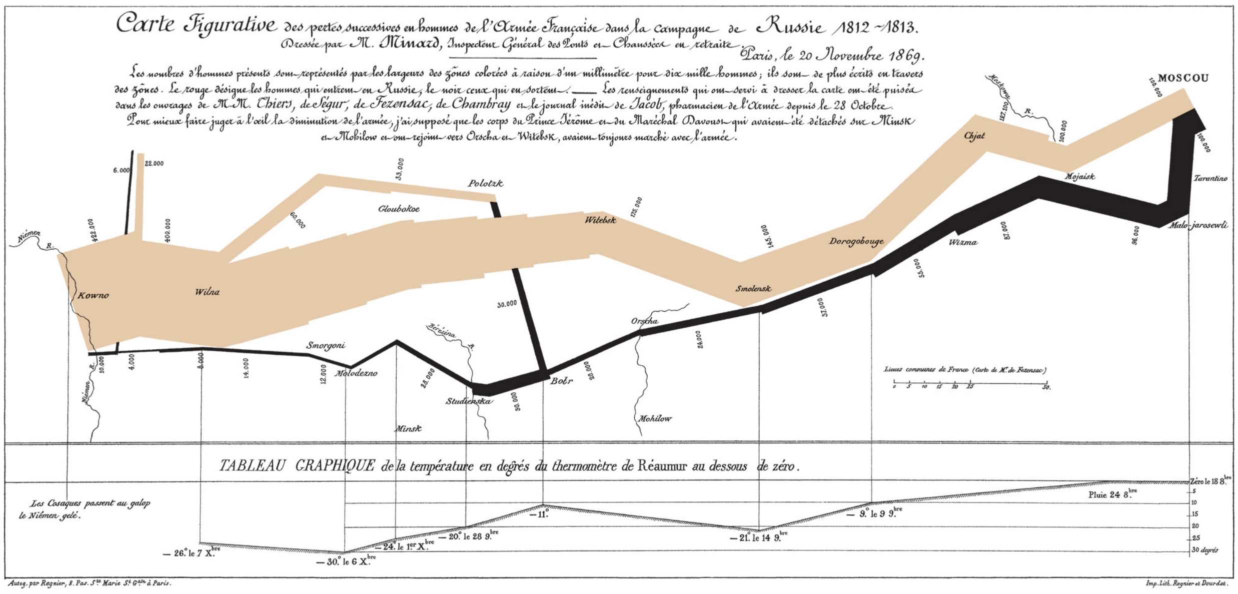
6. A challenge

Exercise. Find an interesting way to depict the action in Game of Thrones. Here is some inspiration.

First, a Sankey diagram by Jeffrey Lancaster (https://medium.com/@jeffrey.lancaster/the-ultimate-game-of-thrones-dataset-a100c0cf35fb). (In the `actions` dataframe, the `locseq` column is ordered according to the y-position in this diagram.)



Second, [Minard's famous plot (https://en.wikipedia.org/wiki/Charles_Joseph_Minard#Work)](https://en.wikipedia.org/wiki/Charles_Joseph_Minard#Work) of Napoleon's march to Russia, and a [ggplot recreation by Andrew Heiss (https://github.com/andrewheiss/fancy-minard)](https://github.com/andrewheiss/fancy-minard).


Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812–1813.
Dressée par M. Minard, Inspecteur Général des Ponts et Chaussées en retraite.
Paris, le 20 Novembre 1869.
Les nombres d'hommes présents sont représentés par les largeurs des zones colorées à raison d'un millimètre pour dix mille hommes; ils sont de plus écrits en travers des zones. Le rouge désigne les hommes qui entrent en Russie, le noir ceux qui en sortent. —— Les renseignements qui ont servi à dresser la carte ont été puisés dans les ouvrages de MM. Thiers, de Ségur, de Fezensac, de Chambray et le journal inédit de Jacob, pharmacien de l'Armée depuis le 28 Octobre.
Pour mieux faire juger à l'œil la diminution de l'armée, j'ai supposé que les corps du Prince Jérôme et du Maréchal Davoust qui avaient été détachés sur Minsk et Mobilow et ont rejoint vers Orscha et Witebsk, avaient toujours marché avec l'armée.
MOSCOU
Kowno
Wilna
Gloubokoe
Polotzk
Witebsk
Smolensk
Dorogobouge
Chjat
Mojaisk
Tarantino
Malo-jarosewli
Wixma
Orscha
Mohilow
Bobr
Studienska
Minsk
Molodezno
Smorgoni
Niemen
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro.
Les Cosaques passent au galop le Niémen gelé.
– 26° le 7 X.bre
– 30° le 6 X.bre
– 24° le 1er X.bre
– 20° le 28 9.bre
– 11°
– 21° le 14 9.bre
– 9° le 9 9.bre
Pluie 24 8.bre
Zéro le 18 8.bre
30 degrés
Lieues communes de France
Autog. par Regnier, 8. Pas. Ste Marie St Gain à Paris.
Imp. Lith. Regnier et Dourdet.