Introduction to ggplot2

This practical introduces a slightly different method of creating plots in R using the ggplot2 package. The package is an implementation of Leland Wilkinson's Grammar of Graphics—a general scheme for data visualization that breaks up graphs into semantic components such as scales and layers. ggplot2 can serve as a replacement for the base graphics in R (the functions you have been plotting with today) and contains a number of default options that match good visualisation practice.

The maps we produce will not be that meaningful—the focus here is on sound visualisation with R and not sound analysis (I know one is useless without the other!). Whilst the instructions are step by step you are encouraged to deviate from them (trying different colours for example) to get a better understanding of what we are doing.

ggplot2 is (in my opinion) one of the best documented packages in R. The full documentation for it can be found here:

http://docs.ggplot2.org/current/

There is also a cookbook for R with some nice examples:

http://wiki.stdout.org/rcookbook/Graphs/

Let's first download the data we need today:

See link on blog.

Unzip it and save it to a new folder.

Open R and load the packages:

library(maptools)
library(ggplot2)
library(gpclib) #this may not be installed

Set your working directory (where your files are saved)

setwd("XX/XX")

It is worth noting that the basic plot() function requires no data preparation but additional effort in colour selection/ adding the map key etc. qplot() and ggplot() (installed in the ggplot2 package) require some additional steps to format the spatial data but select colours and add keys etc automatically. More on this later.

In order to produce a map we need to load the shapefile containing the spatial data and boundaries. This uses the readShapePoly function installed with the maptools package.

Load the shapefile

License: CC-BY-NC-SA
sport<- readShapePoly("london_sport.shp")

All shapefiles have an attribute table. This is loaded with readShapePoly and can be treated in a similar way to a data.frame. This dataset has been taken from here: http://data.london.gov.uk/datastore/package/active-people-survey-participation and combined with the borough boundary geometry from the Ordnance Survey. This is publicly available from here thanks to the OS Opendata Scheme: http://www.ordnancesurvey.co.uk/oswebsite/opendata/. The file contains the borough population and the percentage of the population engaging in sporting activities.

R hides the geometry etc of spatial data unless you print the object (using the print() function). Look at the headings of sport.

names(sport)

As a first attempt with ggplot2 we can create a scatter plot with the data. Type:

p<-ggplot(sport@data, aes(Partic_Per,Pop_2001))

What you have just done is set up a ggplot object where you say where you want the input data to come from (sport@data (the @ enables you to access the attribute table of the sport shapefile)) and what parts of that data frame you wish to use (Partic_Per and Pop_2001), this has to happen within the brackets after aes(). If you just type p and hit enter you get the error “No layers in plot”. This is because you have not told ggplot what you want to do with the data. We do this by adding so-called “geoms” in this case geom_point().

p+geom_point()

Within the brackets you can alter the nature of the points. Try

p+geom_point(colour="red", size=2)

If you want to scale the points by borough population and colour them by sports participation this is also fairly easy by adding another aes().

p+geom_point(aes(colour=Partic_Per, size=Pop_2001))

The real power of ggplot2 lies in its ability to add layers to a plot. In this case we can add text to the plot

p+geom_point(aes(colour=Partic_Per,size=Pop_2001)) +geom_text(size=2,aes(label=name))

So this idea of layers (or geoms) is quite different from the standard plot functions in R, but you will find that each of the functions (see the documentation for a full list) does a lot of clever stuff to make plotting much easier.
The following steps will create a map to show the percentage of the population in each London Borough who regularly participate in sports activities.

To get shapefiles into a format that can be plotted we have to use the `fortify()` function. Spatial objects in R have a number of slots containing the various items of data (polygon geometry, projection, attribute information) associated with a shapefile. The “polygons” slot contains the geometry of the polygons in the form of the XY coordinates used to draw the polygon outline. The generic plot function can work out what to do with these, ggplot2 cannot. We therefore need to extract them as a data frame. This can be done using the `fortify` function written specifically for this purpose. For `fortify` to work you need to activate the gpclib library like this

```r
gpclibPermit()
```

```r
sport_geom <- fortify(sport, region="ons_label")
```

This step has lost the attribute information associated with the sport object. We can add it back using the `merge` function (this performs a data join). To find out how this function works look at `?merge`.

```r
sport_geom <- merge(sport_geom, sport@data, by.x="id", by.y="ons_label")
```

Have a look at the `sport_geom` object to see its contents. You should see a large data frame containing the latitude and longitude (they are actually eastings and northings as the data are in British National Grid format) coordinates alongside the attribute information associated with each London Borough. If you type `print(sport_geom)` you will just how many coordinate pairs are required!

```r
head(sport_geom)
```

It is now straightforward to produce a map using all the built in tools (such as setting the breaks in the data) that ggplot2 has to offer. `coord_equal()` is the equivalent of `asp=T` in regular plots with R:

```r
Map <- ggplot(sport_geom, aes(long, lat, group=group, fill=Partic_Per)) + geom_polygon() + coord_equal() + labs(x="Easting (m)", y="Northing (m)", fill="% Sport Partic.") + ggtitle("London Sports Participation")
```

There is a lot going on in the code above. Take a moment to think about what each of the elements of code above has been designed to do. Also note how the `aes()` components can be combined into one set of brackets after `ggplot`, rather than broken into separate parts as we did above. The different plot functions still know what to do with these. The `group=group` points `ggplot` to the group column added by `fortify()` and it identifies the groups of coordinates that pertain to individual polygons (in this case London Boroughs).
The default colours are really nice but we may wish to produce the map in black and white:

```
Map + scale_fill_gradient(low="white", high="black")
```

This should have produced a map that looks like this:

![London Sports Participation Map](image)

Saving plots is also easy. You just need to type

```
ggsave("my_map.pdf")
```

to get a pdf version of the plot. To get a png type “my_map.png”. You can also edit the dimensions of the map and, for images only, its resolution.

```
ggsave("my_large_plot", scale=3, dpi=400)
```

**Using ggplot2 for Descriptive Statistics**

For this we will use a new dataset:

```
input<-read.csv("ambulance_assault.csv")
```

This contains the number of ambulance callouts to assault incidents (downloadable from the London DataStore) between 2009 and 2011.

Take a look at the contents of the file:

```
head(input)
```

We can now plot a histogram to show the distribution of values.
p_ass<- ggplot(input, aes(x=assault_09_11))

Remember the "ggplot(input, aes(x=assault_09_11))" section means create a generic plot object (called p_ass) from the input object using the assault_09_11 column as the data for the x axis. To create the histogram you need to tell R that that is what you want to go with

p_ass+geom_histogram()

The message

stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.

relates to the binning, if you want the bins (and therefore the bars) to be thinner (ie representing fewer values) you need to make the bins smaller by adjusting the binwidth. Try:

p_ass+geom_histogram(binwidth=10)+geom_density(fill=NA, colour="black")

You can also overlay a density distribution over the top of the histogram. For this we need to produce a second plot object that says we wish to use the density distribution as the y variable.

p2_ass<- ggplot(input, aes(x=assault_09_11, y=..density..))

p2_ass+geom_histogram()+geom_density(fill=NA, colour="red")

What kind of distribution is this plot showing? You can see that there are a few Wards with very high assault incidences (over 750). To find out which ones these are we can select them (this is similar to what we covered in the previous session).

input[which(input$assault_09_11>750),]

It is perhaps unsurprising that St James's and the West End have the highest counts...This plot has provided a good impression of the overall distribution, but what are the characteristics of each distribution within the Boroughs? Another type of plot that shows the core characteristics of the distribution is a box and whisker plot. These too can be easily produced in R (you can't do them in Excel!). We can create a third plot object (note that the assault field is now y and not x):

p3_ass<-ggplot(input, aes(x=Bor_Code,y=assault_09_11))

and convert it to a boxplot

p3_ass+geom_boxplot()

I think this would look a little better flipped round
p3_ass+geom_boxplot()+coord_flip()

Now each of the borough codes can be easily seen. No surprise that the Borough of Westminster (00BK) has the two largest outliers. In one line of code you have produced an incredibly complex plot rich in information. This demonstrates why R is such a useful program for these kinds of statistics.

If you want an insight into some of the visualisations you can develop with this type of data we can do faceting based on the example of the histogram plot above.

p_ass+geom_histogram()+facet_wrap(~Bor_Code)

We need to do a little bit of tweaking to make this plot publishable but I wanted to demonstrate that it is really easy to produce 30+ plots on a single page! Faceting is an extremely powerful way of visualizing multidimensional datasets and is especially good for showing change over time.

Advanced Task: Faceting for Maps

library(reshape)  #this may not be installed. If not skip the next two steps...

Load the data- this shows historic population values between 1801 and 2001 for London, again from the London data store.

london.data <- read.csv("census-historic-population-borough.csv")

“Melt” the data so that the columns become rows.

london.data.melt <- melt(london.data, id=c("Area.Code", "Area.Name"))

Only do this step if reshape and melt failed

london.data.melt<-read.csv("london_data_melt.csv")

Merge the population data with the London borough geometry contained within our sport_geom object.

plot.data <- merge(sport_geom, london.data.melt, by.x="id", by.y="Area.Code")

Reorder this data (ordering is important for plots).

plot.data <- plot.data[order(plot.data$order),]

We can now use faceting to produce one map per year (this may take a little while to appear).
ggplot(data = plot.data, aes(x = long, y = lat, fill = value, group = group)) + geom_polygon() + geom_path(colour="grey", lwd=0.1) + coord_equal() + facet_wrap(~variable)

Again there is a lot going on here so explore the documentation to make sure you understand it. Try out different colour values as well.

Add a title and replace the axes names with “easting” and ‘northing” and save your map as a pdf.