Route Analysis using R and Google Maps

Open up R Studio (click **Start > All Programs > RStudio > RStudio** or double-click the icon on the desktop).

One of the libraries we will use is `ggmap`, which allows us to quickly and easily include some Google Maps within R. First, we need to load the library, so type into the console:

```
library(ggmap) # use install.packages("ggmap") if not already installed
```

This will also load various other libraries and you will probably see information messages. We can then create a quick map of Leeds with the `qmap()` function. You should see the map appear in the 'Plots' window to the left of the R Studio window.

```
qmap('Leeds')
```

We can also search for anything we would search for in the Google Maps search box - e.g. different locations, or postcodes. For example, try:

```
qmap('L69 3BX')
```

Try different postcodes or locations to search for, and see what happens. You will see that the scale of the map stays the same - this is fine for Leeds, but of limited use for somewhere smaller (like Norwich) or larger (like Ireland). The default zoom is level 10, but we can change this to something more useful. Higher numbers zoom in more (i.e. show a smaller area). Try:

```
qmap('LS2 9JT', zoom = 17)
```

This shows us the map centred on the Parkinson building (although it isn't marked on the map).

As well as the maps layer, we can access the satellite and hybrid map types from Google Maps:

```
qmap('LS2 9JT', zoom = 17, maptype = 'satellite')
```

One useful shortcut when using R Studio (or R on its own) is that pressing the 'up' key on the keyboard will show your previous command. You can then edit this and press return to run it, which avoids the need to type the whole command out again! Try it with the command below:

```
qmap('LS2 9JT', zoom = 17, maptype = 'hybrid')
```

Routing

Just as you can get directions from the Google Maps website, you can also access the same tools through the API. The code below passes an origin and destination to the routing command and then shows the route on a map. Try running this code, and then try changing to origin and/or destination and try running it again. You may need to adjust the location of the map to show the whole route.

With a bit of extra R code, we can create a data frame of different origin and destination pairs, and then write the code to get R to look through the API and produce a route for each of these.

First of all, add the code to get R to save the map as a PDF. Run this code, and R will save the map as a PDF file in your working folder. (Run `getwd()` if you are not sure where your working folder is).

```
#save as PDF
pdf(file="image.pdf")
  from <- 'Leeds station, New Station Street, Leeds LS1 5DL, United Kingdom'
  to <- 'LS2 9JT'
  route_df <- route(from, to, structure = 'route', mode = 'walking')
  qmap('Merrion Centre', zoom = 15) +
    geom_path(
      aes(x = lon, y = lat),  colour = 'red', size = 1.5,
      data = route_df, lineend = 'round')
#stop saving as PDF
dev.off()
```

Then we can add a data frame and a loop, to run the code for each entry.

```
#setup variable with list of origins
  origins <- c("Leeds station, New Station Street, Leeds LS1 5DL","Leeds","Manchester",
               "Manchester Oxford Road")
  destinations <- c("LS2 9JT","Manchester","Liverpool","Manchester Picadilly")
  map_center <- c("Merrion Centre","Marsden, UK","Warrington","Sackville St, Manchester")
  zoom_level <- c(15,10,10,16)
#combine origins and destinations to create a data frame
```

```
  df_orig_dest <- data.frame(origins,destinations,map_center,zoom_level)
#loop through for each map
for (i in 1:length(df_orig_dest)) {
  #setup file name
  filename <- paste0("route_",df_orig_dest[i,1],"_to_",df_orig_dest[i,2],".pdf")
  #save as PDF and set file name
  pdf(file=filename)
    from <- as.character(df_orig_dest[i,1]) #check whether this is needed
    to <- as.character(df_orig_dest[i,2])
    route_df <- route(from, to, structure = 'route', mode = 'walking')
    print(qmap(as.character(df_orig_dest[i,3]), zoom = df_orig_dest[i,4]) +
      geom_path(
        aes(x = lon, y = lat),  colour = 'red', size = 1.5,
        data = route_df, lineend = 'round'))
  #stop saving as PDF
  dev.off()
#end loop
}
```

Try experimenting with different parts of the code, to, for example, change the colour for each route, or add more entries. You can also save the maps as images - for example, type `?jpeg` into the console to see how to get the map as a JPEG.

Routing with CycleStreets.net

The routes from Google are useful, containing data on distance and time for different modes of transport. However, they are limited: we are only given some of the route, which is not allocated to the route network, only containing nodes where the route diverges.

To overcome this problem, we can use alternative route finding services such as https://graphhopper.com/ and http://project-osrm.org/. CycleStreets.net is a routing service designed by cyclists for cyclists that we will use to illustrate routing using other external services in R. It has various advantages over the `route()` function described above but you need a CycleStreets.net 'API key' for it to work. Apply here: http://www.cyclestreets.net/api/ .

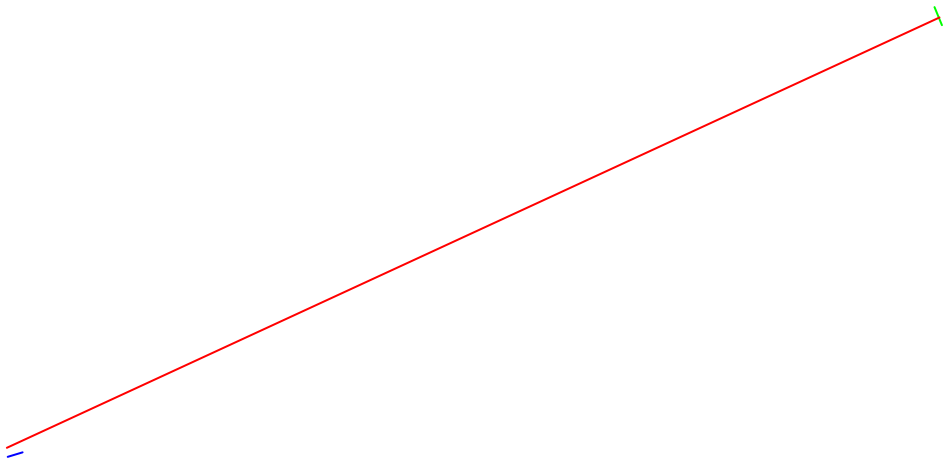
For a project funded by the UK's Department for Transport (DfT), an R interface to the CycleStreets.net routing service has been created. This is included in the R package **stplanr**. Let's download this package to reproduce the routes using CycleStreet.net.

```
library(sp) # enables spatial data classes
library(devtools) # to install packages from github
install_github("robinlovelace/stplanr") # install the package
library(stplanr)
cckey <- "f3fe3d078ac34999" # add your API key - will be vary
```

Convert the OD matix into straight lines

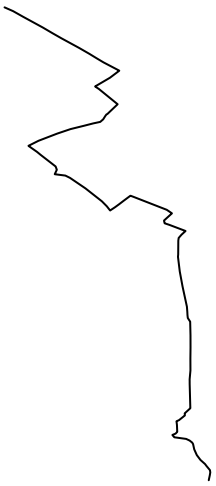
To convert the text string origins and destinations into routes, we first must convert them into objects of the `SpatialLinesDataFrame` data class:

```
f <- df_orig_dest[-3,]
coords <- geocode(c(origins, destinations))
cents <- SpatialPointsDataFrame(coords = as.matrix(coords),
  data = data.frame(code = c(origins, destinations)))
rlines <- gFlow2line(flow = f, zones = cents)
plot(rlines, col = c("green", "red", "blue", "black"))
```



We have successfully saved our 'desire lines'. Now let's convert them to navigable cycle routes and plot them:

```
rquiet <- gLines2CyclePath(l = rlines, plan = "quietest")
plot(rquiet[1,]) # route from Leeds station to Leeds University (North - South)
```



```
plot(rquiet[2,]) # route from Leeds to Manchester!
```