Data Science: Visualisations in R - Assignment 5

Github of Alex Hanna and Robin Love Lace, Healy (2018) Data visualization, Chapter 7

dr. Mariken A.C.G. van der Velden

November 29, 2019

Instructions for the tutorial

Read the *entire* document that describes the assignment for this tutorial. Follow the indicated steps. Each assignment in the tutorial contains several exercises, divided into numbers and letters (1a, 1b, 1c, 2a, 2b, etc.).

Write down your answers with **the same number-letter combination** in a separate document (e.g., Google Doc), and submit this document to Canvas in a PDF format.

Note: this is the first time that this course has ever been taught. Because of rapidly changing techniques, the course tries to keep up/ reflect those changes. As a result, the assignments may contain errors or ambiguities. If in doubt, ask for clarification during the hall lectures or tutorials. All feedback on the assignments is greatly appreciated!

Relational Data & Networks

It's rare that a data analysis involves only a single table of data. Typically you have many tables of data, and you must combine them to answer the questions that you're interested in. Collectively, multiple tables of data are called *relational data* because it is the relations, not just the individual datasets, that are important.

Relations are always defined between a pair of tables. All other relations are built up from this simple idea: the relations of three or more tables are always a property of the relations between each pair. Sometimes both elements of a pair can be the same table! This is needed if, for example, you have a table of people, and each person has a reference to their parents.

Social network analysis (SNA) is a body of methods that deals with relational data. SNA, also called network analysis, and the related fields of network science and graph theory, is a body of methods used to study the relationship between entities. Entities can be individuals, organizations, Twitter users, countries, or a mix of any of the above. Network analysis distinguishes itself from other types of analysis because the focus is on the relationship between entities rather than focusing solely on the properties of the entity itself.

Typically, we want to use network analysis if we have a sense that the relationships or transactions between actors/entities are the most critical part of the story. If you have a sense that the interlocking connection between entities is more important than their individual attributes, or if networked connections between entities forces an endogeneity problem which cannot be resolved by more conventional quantitative regression techniques, then network analysis may be a good approach for you.

Although network analysis as is presented today is sometimes synonymous with computational social science or "big data" methods, network analysis has a long history within sociology and political science. Some of the more innovative uses of network analysis focus on uses that work with archival and historical data.

In a classic article, Padgett and Ansell discuss the rise of the Medici family in 14th-century Florence, Italy. While many families in Florence were attempting to accumulate power and influence within this Renaissance state, the Medici family was able to do so in such a way that outpaced all others. Padgett and Ansell illustrate how they were able to do this using network methods. They used a method called blockmodeling to simplify multiple network relationships and positions to get at an underlying network structure. Padgett and Ansell found that the Medicis were able to consolidate power by positioning themselves as a critical broker in marriage and economic (trade, partnership, and real estate) networks within the larger group of elite Florentine families.

It illustrates a highly complicated network based on data which are over six centuries old. The **actor or entity** here is the elite family. Second, there is not a single type of relationship in this network – this is a **multiplex** network, meaning there are multiple types of relationships which exist between entities. Relationships are both **undirected** and **directed**, which means they are not all mutual. More on that below. Furthermore, what this means is that the **medium** of what travels across the network tie is different for each network. It is a combination of trust, cooperation, and resources. Third, the data are archival and based upon historical research. The network must be constructed and operationalized explicitly, rather than something like a Twitter retweet, which tends to be accepted somewhat uncritically.

Compare that to this classic network from Adamic and Glance on the political blogosphere circa 2005. This may have been the first of many articles which illustrated the growing political polarization in US political life. The actor here is a political blog. The relationship here is the hyperlink between one blog and another. Because links by their nature go from one blog to another, this is a directed network. The links are not necessarily mutual.

In this network visualization, **colors** and **size** take on a particular type of meaning. Red actors signify conservative blogs, while blue actors signify liberal ones. Furtermore, red lines represent conservative-to-conservative links, and blue liberal-to-liberal. Orange lines, however, represent cross-ideological relationships. Lastly, size indicates how many links are coming into a particular blog. Those colors and sizes indicate something about the actors themselves; they are **attributes** of the actors.

In the former example, we used network analysis to focus on the *centrality of a particular actor* – the Medici family. However, in the latter example, we used network analysis to focus on the *structure of the network as a whole* – namely how between-ideology linkage is much less common than cross-ideology linkage. Both of these could not be achieved by looking at each entity on its own or entities as an aggregate.

The theoretical groundings of network analysis are scattered around various disciplines of social science, but Emirbayer argues most forcefully for its necessary in social science research. Emirbayer poses *relational* or *transactional* analysis as more ontologically sound than *variable-based* or *substantialist* analysis in the social sciences. The focus should be on the interaction between entities, rather than their properties. Individuals may be said to contain attributes (e.g. gender, race, sexual orientation), but a transactional view would indicate that those attributes are all relational (e.g. Desmond and Emirbayer's discussion of racial domination).

Today, we will be covering the basic and intermediate topics in network analysis. We will cover basic network terminology and concepts, network visualization, and descriptive metrics which describe whole networks, individual nodes, and subgroups. We will also cover statistical inference on determinants of network structure.

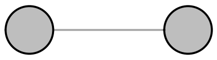
To work with SNA in R, we're going to use two packages:

```r
library(igraph)
library(repr) #for resizing plots
```

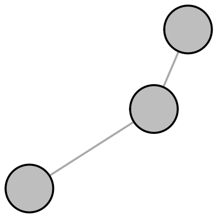
Let's get to defining some terms before we go on, so that we're on the same page. So far I've been using the terms entities or actor to talk about individuals in network analysis. From now on, I will more often use the term node to discuss the entity who is part of the relationship. The terms actor or vertex are synonyms for this.

The connection between two nodes is called an edge (or arc, link, or relation). A network with two nodes and a single edge is called a dyad. A network with three nodes, with any type of configuration of edges between them, is called a triad.

```r
## a dyad
## using lgl layout so that the dyad lays flat
plot(graph_from_literal(A-B), vertex.color = "gray",
     vertex.label = NA, vertex.size = 60,
     layout=layout_with_lgl)
```

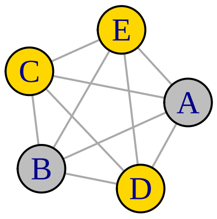


```
## a triad
plot(graph_from_literal(A-B-C), vertex.color = "gray",
     vertex.label = NA, vertex.size = 60,
     layout=layout_nicely)
```



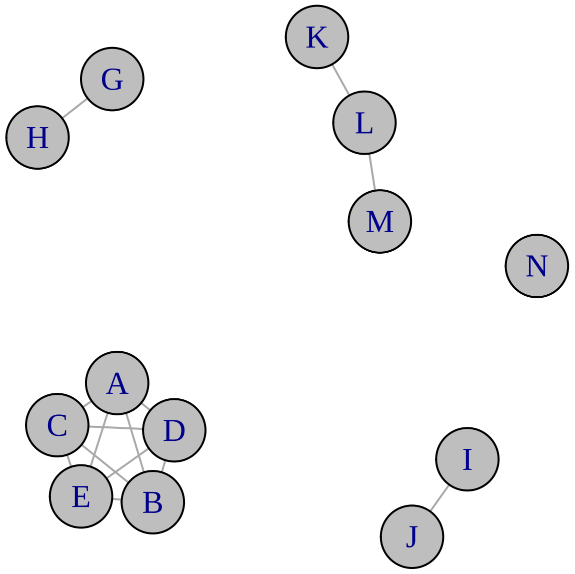
You'll see that we're using a funny little function to draw these networks above called `graph_from_literal`. This lets me literally draw some basic networks using a rudimentary simple syntax. What I want to draw attention to is the name of the function, namely the first word: **graph**. A **graph** is another name for a network and is a term much more common on computer science. A **subgraph** is any subset of a graph. In the network below, the nodes B, C, and D (highlighted in gold) form a subgraph of the larger graph.

```
net <- graph_from_literal(A:B:C:D:E - A:B:C:D:E)
V(net)$color <- c('gray', 'gray', 'gold', 'gold', 'gold')
plot(net, vertex.size = 60)
```



A **component** is a subgraph which is connected together. In the plot below, nodes A through E are a component. Nodes G-H, I-J, K-L-M, and N are components. The largest component is called the **major component** while the others are called **minor components**. N is a special kind of component which is by itself and thus called an **isolate**.

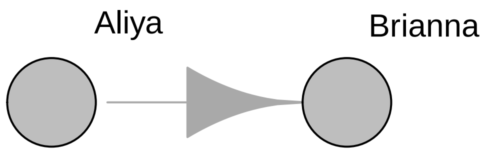
```
options(repr.plot.width = 8, repr.plot.height = 4)
net <- graph_from_literal(A:B:C:D:E - A:B:C:D:E,
                          G-H, I-J, K-L-M, N)
plot(net, vertex.color = 'gray', vertex.size = 25)
```



The edges of a network tend to have a set of common properties. Let's begin with properties of edges. As noted in the Medici example, edges in a network can be either **directed** or **undirected**. The most readily available example we can draw on is from social media – on Facebook, your friendships are mutual. This is an undirected network. Aliya is friends with Brianna and vice versa. On Twitter, however, follower networks are asymmetrical. Aliya may follow Brianna, but Brianna doesn't follow Aliya. This is a directed network. Directed networks are denoted with an arrow in a visualization.

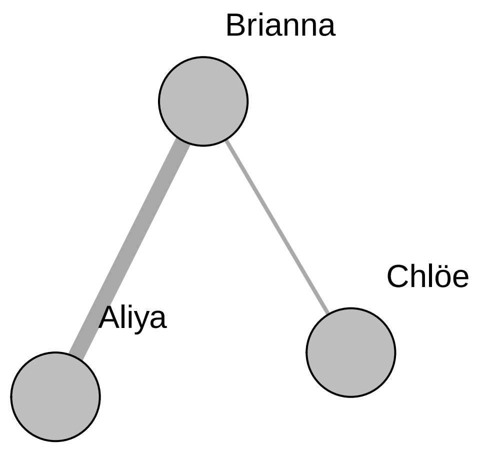
```
# shrink plot size
options(repr.plot.width = 4, repr.plot.height = 3)

net <- graph_from_literal(Aliya --+ Brianna)
plot(net,
     vertex.color = "gray",
     vertex.size = 60,
     vertex.label.color = "black",
     vertex.label.family = "Helvetica",
     vertex.label.dist = 10,
     edge.arrow.size = 3,
     layout=layout_with_lgl)
```



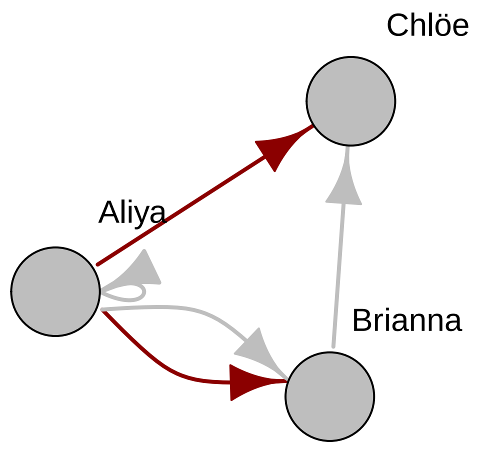
Edges can be **weighted** (valued) or **unweighted** (unvalued), which means they can denote some kind of attribute, such as strength or distance. A Facebook friend in which there's a lot of interaction a good deal of interaction may have a higher weight than one without. Visually, this can be represented in a number of ways. Below, it's denoted with edge width. We'll get into that more in the next module.

```
net <- graph_from_literal(Aliya - Brianna - Chlöe)
E(net)$width <- c(8, 2)
plot(net,
     vertex.color = "gray",
     vertex.size = 60,
     vertex.label.color = "black",
     vertex.label.family = "Helvetica",
     vertex.label.dist = 10)
```



Networks can also be **multiplex**, meaning they can denote multiple relationships. Again, this was the case with the Medici example. Furthermore, nodes can link to themselves, what is called a **self-loop**. Think about retweets – people on Twitter can retweet others as well as retweeting themselves. In the graph below, imagine that retweets are the gray network and @replies are dark red network. Aliya is retweeting and replying to Brianna and Chlöe. Aliya also retweets herself. Brianna retweets Chlöe.

```
net <- graph_from_literal(Aliya -+ Brianna -+ Chlöe,
                          Aliya -+ Brianna, Aliya -+ Chlöe,
                          Aliya-+Aliya, simplify = FALSE)
plot(net,
     edge.color = c("dark red", "grey", "grey", "dark red", "grey"),
     vertex.color = "gray",
     vertex.size = 60,
     edge.width = 2,
     edge.arrow.size = 1.5,
     vertex.label.color = "black",
     vertex.label.family = "Helvetica",
     vertex.label.dist = 10)
```

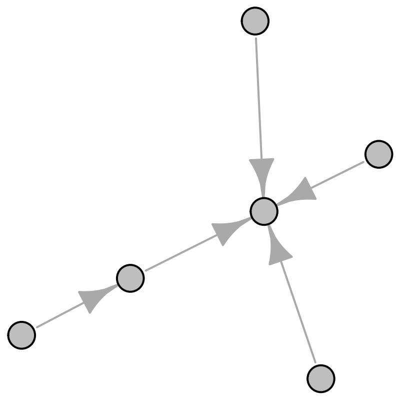


Lastly, networks can change over time. They can be either static or dynamic. We won't really touch on these much in this workshop, so we'll avoid the plotting.

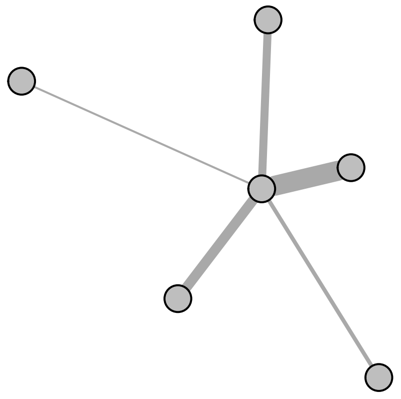
Assignment 1: Basic Understanding

1a) The following code generates two random networks (called a preferential attachment network). Based on these visualization, can you determine whether the networks are directed or undirected? Weighted or unweighted? Singular or multiplex?

```r
net1 <- sample_pa(6)
plot(net1, vertex.label = NA, vertex.color = "gray")
```



```r
net2 <- sample_pa(6, directed = FALSE)
E(net2)$weight <- c(10,4,1,2,5)
E(net2)$width <- E(net2)$weight
plot(net2, vertex.label = NA, vertex.color = "gray")
```



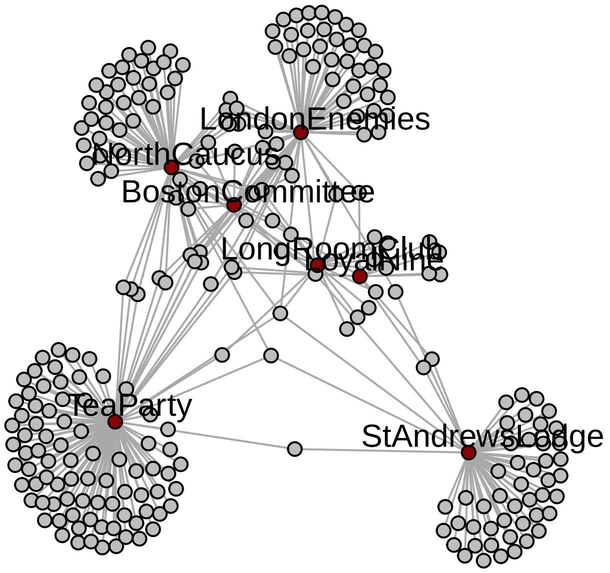
So far we've only discussed networks in which there's a node of a particular type. However, in political and social networks we often observe networks of more than one type of entity. The simplest advancement of this is the **two-mode network** or **bipartite network**, that is, a network where there's two types of nodes.

Two-mode network analysis is most common in the case of individuals and groups, or individuals and events. Take this example from Kieran Healy and using metadata to identify Paul Revere. The data here is a matrix which has on its rows individuals and on its columns whether or not the person was a member of a particular American revolutionary organization.

```r
(df<- read.csv("PaulRevereAppD.csv", row.names = 1))
```

The visualization of bipartite graph can be helpful in illustrating common membership in organizations.

```r
bipartite_revere <- graph.incidence(df)
plot(bipartite_revere,
     vertex.color = ifelse(V(bipartite_revere)$type, "dark red", "gray"),
     vertex.size = 5,
     vertex.label = ifelse(V(bipartite_revere)$type, V(bipartite_revere)$name, NA),
     vertex.label.dist = 1,
     vertex.label.family = "Helvetica",
     vertex.label.color = "black")
```



However, a major insight of these types of networks highlighted by Ron Breiger is the two-mode networks have a duality – they can be transformed into one-mode networks (e.g. networks with only one type of node) such that we can highlight the importance of one mode in terms of the other, or vice versa. In terms of the Paul Revere dataset, we can see the importance of the individuals based on their group memberships, or we can see the importance of the groups based on the individual memberships.

The transformation is more or less this: if we have an matrix like the dataset above, if we want to obtain the one-mode network of the first mode, we multiple the matrix with its transpose:

$$G_1 = A(A^T)$$

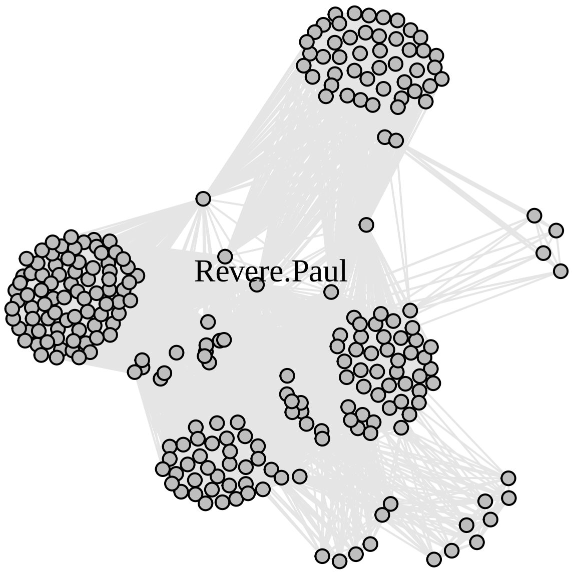
Conversely, to obtain the one-mode network of the second mode, we multiple the transpose of the matrix with the matrix:

$$G_2 = (A^T)A$$

```
df <- as.matrix(df) ## only matrices can be transposed

person_net <- df %*% t(df) # A*A^T
diag(person_net) <- NA
revere_person_g <- graph.adjacency(person_net,mode="undirected", weighted=NULL, diag=FALSE)

## gets rid of multiplex links and loops
revere_person_g <- simplify(revere_person_g)
plot(revere_person_g,
     vertex.color = "grey",
     vertex.size = 5,
     vertex.label = ifelse(V(revere_person_g)$name == "Revere.Paul", V(revere_person_g)$name, ""),
     vertex.label.dist = 1,
     edge.color = "grey90",
     vertex.label.color = "black")
```



Paul Revere himself is highlighted to illustrate his importance to the connectivity of this network. While we lose some information in terms of group membership, we gain a good deal in highlighting the importance of relationships in the mode we may be more interested in.

1b) The syntax for creating undirected graphs with `graph_from_literal` is `A - B`. Create an undirected graph in which Amsterdam is connected to Rotterdam and Eindhoven, and Rotterdam is connected to Eindhoven.

1c) Generate and plot the one-mode network for the groups by filling out the blanks below. What do you notice?

```
revere.group.net <- ____ %*% ____
diag(revere.group.net) <- NA
revere.group.g <- graph.adjacency(____,
                                  weighted = TRUE,
                                  mode="undirected",
                                  diag=FALSE)

plot(revere.group.g,
     vertex.color = "grey",
     vertex.size = 5,
     vertex.label.dist = 1,
     edge.color = "grey",
     edge.width = E(revere.group.g)$weight,
     vertex.label.color = "black")
```

We turn our attention to the various methods of gathering network data. Our minds may go automatically to web scraping or various APIs for network data. But there are many different methods of gathering network data. The first is **digital** – accessing network data through web scraping (such as Adamic and Glance's hyperlink network), the Facebook Graph API (which has graph in the name), and the Twitter API. These data appear to be "born networked" – it doesn't take much of a stretch of the imagination to ask how they are relational. However, one should definitely ask what the nature of that network relation is. Boyd, Golder, and Lotan, for instance, ask about the different conversational aspects of retweets and what that relationship means to different people.

The next most popular which we tend to see is **survey-based**, which ask about networks of individuals with whom individuals correspond or interact. Much of the research in the past 40 years on networks has been based on surveys which ask about core discussion networks – close networks of people with whom we discuss important topics. The General Social Survey in the US has asked about this for years and has been a subject of contention.

Thirdly, network ties can be inferred via qualitative methods, namely, **interviews and ethnographies**. These methods attempt to gather information on social ties by interacting and/or observing people. A study by Matt Desmond discusses "disposible ties" – intensely strong, yet temporary ties amongst the urban poor. Ethnographical methods often discuss these ties in network analysis language without formalizing the relations between individuals.

Lastly, **archival** methods draw networks from historical and archival sources. We have already seen an example about with the Medici network. There are many examples from within historical sociology, including Bearman et al.'s methodological article on historical casing from first-person narratives and Mohr and Duquenne's exploration of the turn-of-the-century language around the deserving and undeserving poor.

Before moving on, I want to say a word about **explicit vs. implicit networks**. There are some things which seem to be explicitly referred to as networks in the above methods – Twitter retweets, survey name generators. However, there's a good deal of data which has an implicit network structure which has to be specified by the researcher. Examples of these include textual networks – which is an alternative way of performing text analysis – hashtag cooccurrence analysis, and legislative cosponsorship data. While these do not appear to be data sources where you can apply network data, they turn out to be places where thinking about things in a relational manner ends up being very helpful.

There's three main ways that networks are represented and can be loaded into R. There are several other properitary formats which I will not discuss here. These, however, are the main ways which we can read in networks to `igraph` and other R packages.

```r
## incidence matrix
head(df)
```

```
##                      StAndrewsLodge LoyalNine NorthCaucus LongRoomClub
## Adams.John                        0         0           1            1
## Adams.Samuel                      0         0           1            1
## Allen.Dr                          0         0           1            0
## Appleton.Nathaniel                0         0           1            0
## Ash.Gilbert                       1         0           0            0
## Austin.Benjamin                   0         0           0            0
##                      TeaParty BostonCommittee LondonEnemies
## Adams.John                  0               0             0
## Adams.Samuel                0               1             1
## Allen.Dr                    0               0             0
## Appleton.Nathaniel          0               1             0
## Ash.Gilbert                 0               0             0
## Austin.Benjamin             0               0             1
```

Incidence matrix - The incidence matrix makes the most sense as a representation for two-mode networks, since it is matching one class of entities to another. In the Paul Revere example, the people are on the rows and the organizations are on the columns. If the network is unweighted, then for all nodes which have a connection, the corresponding cell is 1. For all other cells, the value is 0. If it network is weighted, then instead of 1, the value is the weight.

```r
## adjacency matrix
revere_person_df <- data.frame(as.matrix(as_adjacency_matrix(revere_person_g)))

revere_person_df[1:5,1:5] #try out head(revere_person_df)
```

```
##                      Adams.John Adams.Samuel Allen.Dr Appleton.Nathaniel
## Adams.John                0             1         1                   1
## Adams.Samuel              1             0         1                   1
## Allen.Dr                  1             1         0                   1
## Appleton.Nathaniel        1             1         1                   0
## Ash.Gilbert               0             0         0                   0
##                   Ash.Gilbert
## Adams.John                 0
## Adams.Samuel               0
## Allen.Dr                   0
## Appleton.Nathaniel         0
## Ash.Gilbert                0
```

Adjacency matrix - An adjacency matrix operates on effectively the same principle as the incidence matrix for one-mode networks. The adjacency matrix is symmetric across the diagonal for undirected networks but not for directed networks. The code above converts it to a data frame for ease of presentation.

```
## edgelist
head(revere_person_el <- data.frame(as_edgelist(revere_person_g)))
```

```
##          X1                 X2
## 1 Adams.John       Adams.Samuel
## 2 Adams.John           Allen.Dr
## 3 Adams.John Appleton.Nathaniel
## 4 Adams.John       Ballard.John
## 5 Adams.John   Barber.Nathaniel
## 6 Adams.John         Bass.Henry
```

Edgelist - This is the simplest data format. It is a simple list of edges in the graph. Weights will often be denoted by a third column. Again, the code above converts the edgelist to a data frame for ease of use.

`igraph` (documentation) is a robust and fast network package developed by Gabor Csardi. We are going to deal with for the majority of this workshop, although there are a number of supplemental packages which are great for plotting and doing inference. We'll introduce them as needed. The good news is that `igraph` plays nicely with all the other network packages.

`igraph` has a class of method which can create an network object from many different formats. We've already seen one in `graph_from_literal`. We also have the following:

- `graph_from_edgelist`
- `graph_from_adjacency_matrix`
- `graph_from_incidence_matrix`
- `graph_from_data_frame` - creates a graph from an R data frame type

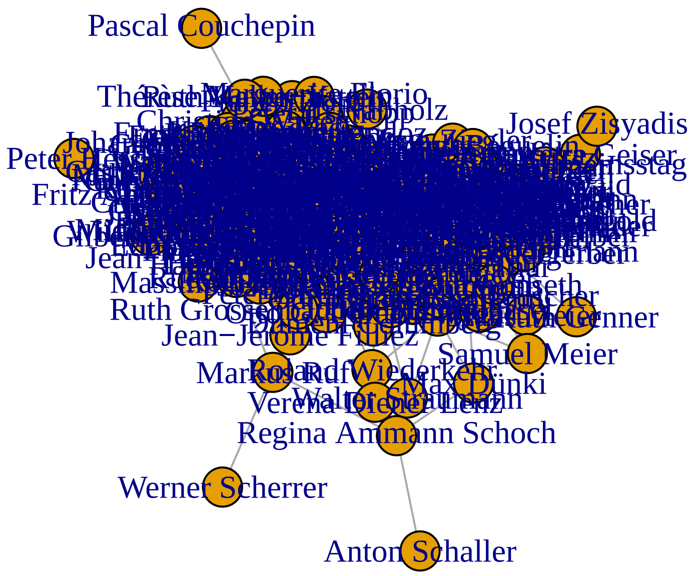
1d) Consider a data set of tweets with information on retweets and @mentions. How do these relationships differ from each other? How are they the same?

1e) Let's load up a new dataset – the cosponsorship network for Conseil National of the Swiss Parliament, from 1995-1999. This data is part of a larger database of cosponsorship data created by Françoise Briatte. This data has been converted from gexf format, and it has been converted from a directed to an undirected graph. Use `head()` to answer the question what kind of data source this is? What kind of data format is this?

```
data_conseil = read.csv('net_ch_cn1995-1999.csv', header = TRUE, as.is = TRUE)
```

1f) Complete the following code to create the network which is appropriate to this data format.

```
plot(graph_from_edgelist(as.matrix(data_conseil), directed = FALSE))
```



Other type of Relations: Spatial Sata

R has a huge and growing number of spatial data packages. We recommend taking a quick browse on R's main website to see the spatial packages available.

In this tutorial we will use the following packages:

- **ggmap**: extends the plotting package `ggplot2` for maps
- **rgdal**: R's interface to the popular C/C++ spatial data processing library `gdal`
- **rgeos**: R's interface to the powerful vector processing library `geos`
- **maptools**: provides various mapping functions
- **dplyr** and **tidyr**: fast and concise data manipulation packages
- **tmap**: a new packages for rapidly creating beautiful maps

```
x <- c("ggmap", "rgdal", "rgeos", "maptools", "tidyverse", "tmap")
#install.packages(x) # warning: uncommenting this may take a number of minutes
lapply(x, library, character.only = TRUE) # load the required packages
```

The first file we are going to load into R Studio is the `london_sport` shapefile. The data can be downloaded from Robin Love Lace's Github page. It is worth looking at this input dataset in your file browser before opening it in R. You will notice that there are several files named "london_sport"", all with different file extensions. This is because a shapefile is actually made up of a number of different files, such as .prj, .dbf and .shp. You could also try opening the file"london_sport.shp" file in a conventional GIS such as QGIS to see what a shapefile contains. Once you think you understand the input data, it's time to open it in R. There are a number of ways to do this, the most commonly used and versatile of which is readOGR. This function, from the `rgdal` package, automatically extracts the information regarding the data. `rgdal` is R's interface to the "Geospatial Abstraction Library (GDAL)" which is used by other open source GIS packages such as QGIS and enables R to handle a broader range of spatial data formats.

```
lnd <- readOGR(dsn = "data", layer = "london_sport")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/Users/velden/surfdrive/Shared/Teaching/Data Science_Visualization and Analytics/Assignments
## with 33 features
## It has 4 fields
## Integer64 fields read as strings:  Pop_2001
```

`readOGR` is a function which accepts two arguments: `dsn` which stands for "data source name" and specifies the directory in which the file is stored, and `layer` which specifies the file name (note that there is no need to include the file extention .shp). The arguments are separated by a comma and the order in which they are specified is important. You do not have to explicitly type `dsn =` or `layer =` as R knows which order they appear, so `readOGR("data", "london_sport")` would work just as well. For clarity, it is good practice to include argument names when learning new functions so we will continue to do so. The file we assigned to the `lnd` object contains the population of London Boroughs in 2001 and the percentage of the population participating in sporting activities. This data originates from the Active People Survey. The boundary data is from the Ordnance Survey. For information about how to load different types of spatial data, see the help documentation for `readOGR`. This can be accessed by typing `?readOGR`. For another worked example, in which a GPS trace is loaded, please see Cheshire and Lovelace (2014).

Spatial objects like the `lnd` object are made up of a number of different slots, the key slots being `@data` (non geographic attribute data) and `@polygons` (or `@lines` for line data). The data slot can be thought of as an attribute table and the geometry slot is the polygons that make up the physcial boundaries. Specific slots are accessed using the `@` symbol. Let's now analyse the sport object with some basic commands:

```r
head(lnd@data, n = 2)
mean(lnd$Partic_Per)
```

Now we have seen something of the structure of spatial objects in R, let us look at plotting them. Note, that plots use the geometry data, contained primarily in the `@polygons` slot.

```r
plot(lnd)
```

`plot` is one of the most useful functions in R, as it changes its behaviour depending on the input data (this is called *polymorphism* by computer scientists). Inputting another object such as `plot(lnd@data)` will generate an entirely different type of plot. Thus R is intelligent at guessing what you want to do with the data you provide it with.

R has powerful subsetting capabilities that can be accessed very concisely using square brackets,as shown in the following example:

```r
# select rows of lnd@data where sports participation is less than 15
lnd@data[lnd$Partic_Per < 15, ] # we don't use the tidyverse verb select,
```

```
##    ons_label           name Partic_Per Pop_2001
## 17      00AQ         Harrow       14.8   206822
## 21      00BB         Newham       13.1   243884
## 32      00AA City of London        9.1     7181
```

```r
# because it doesn't work well with polygons
```

The above line of code asked R to select only the rows from the `lnd` object, where sports participation is lower than 15, in this case rows 17, 21 and 32, which are Harrow, Newham and the city centre respectively. The square brackets work as follows: anything before the comma refers to the rows that will be selected, anything after the comma refers to the number of columns that should be returned. For example if the data frame had 1000 columns and you were only interested in the first two columns you could specify `1:2` after the

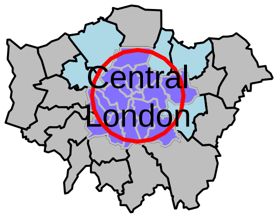
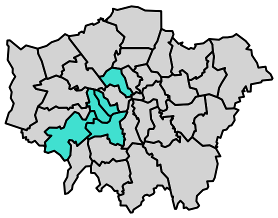


Figure 1: Zones in London whose centroid lie within 10 km of the geographic centroid of the City of London. Note the distinction between zones which only touch or 'intersect' with the buffer (light blue) and zones whose centroid is within the buffer (darker blue).

comma. The : symbol simply means "to", i.e. columns 1 to 2. Try experimenting with the square brackets notation (e.g. guess the result of `lnd@data[1:2, 1:3]` and test it). So far we have been interrogating only the attribute data slot (`@data`) of the `lnd` object, but the square brackets can also be used to subset spatial objects, i.e. the geometry slot. Using the same logic as before try to plot a subset of zones with high sports participation. Try to replicate the following graph:

```r
plot(lnd, col = "lightgrey") # plot the london_sport object
sel <- lnd$Partic_Per > 25
plot(lnd[ sel, ], col = "turquoise", add = TRUE)
```



You have just interrogated and visualised a spatial object: where are areas with high levels of sports participation in London? The map tells us. Do not worry for now about the intricacies of how this was achieved.

As a bonus stage, select and plot only zones that are close to the centre of London. Programming encourages rigorous thinking and it helps to define the problem more specifically:

Assingment 2: Working with Maps

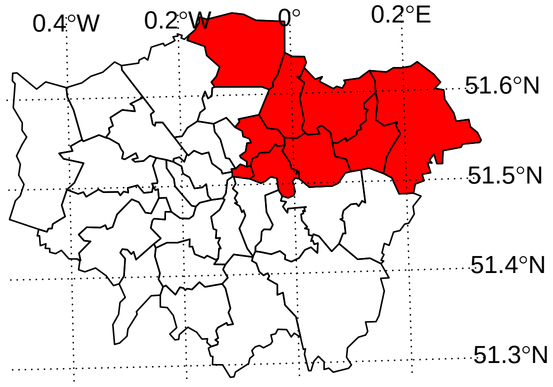
2a) Select all zones whose geographic centroid lies within 10 km of the geographic centroid of inner London.[1]

[1]To see how this map was created, see the code in README.Rmd. This may be loaded by typing `file.edit("README.Rmd")` or online at the underlineREADME page.

The code below should help understand the way spatial data work in R.

```r
# Find the centre of the london area
easting_lnd <- coordinates(gCentroid(lnd))[[1]]
northing_lnd <- coordinates(gCentroid(lnd))[[2]]
# arguments to test whether or not a coordinate is east or north of the centre
east <- sapply(coordinates(lnd)[,1], function(x) x > easting_lnd)
north <- sapply(coordinates(lnd)[,2], function(x) x > northing_lnd)
# test if the coordinate is east and north of the centre
lnd$quadrant <- "unknown" # prevent NAs in result
lnd$quadrant[east & north] <- "northeast"
```

2b) Based on the the above code as reference try and find the remaining 3 quadrants and colour them. Hint - you can use the **`llgridlines`** *function in order to overlay the long-lat lines. Try to desolve the quadrants so the map is left with only 4 polygons.*



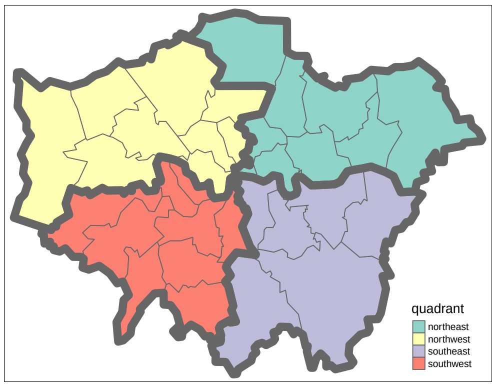


Figure 2: The 4 quadrants of London and dissolved borders. Challenge: recreate a plot that looks like this.

2c) As an alternative to maps, we can consider bins *to visually present spatial data, using the package* `statebins` *(documentation) developed by Bob Rudis. We will use it to look at US state-level election results, (load the* `socviz` *package to access the data set* `election`*.). Statebins is similar to* `ggplot` *but has a slightly different syntax from the one we're used to. It needs several arguments including the basic data frame (the* `state_data()` *argument), a vector of state names (*`state_col`*), and the value being shown (*`value_col`*). In addition, we can optionally tell it the color palette we want to use and the color of the text to label the state boxes. For a continuous variable, like the percentage of votes for Trump or Clinton (*`pct_trump` *and* `pct_clinton`*), we can use* `statebins_continuous()`*. Try to replicate the following plots:*

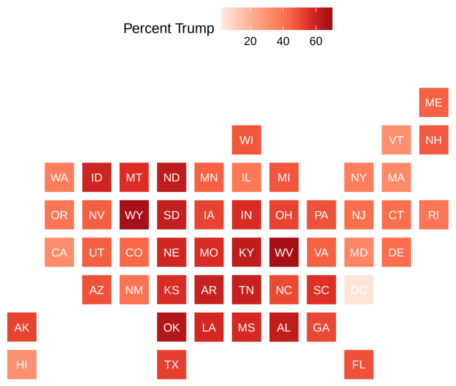


Figure 3: Statebins of the election results. DC is omited from the Clinton map to prevent the scale becoming unbalanced. Challenge: recreate a plot that looks like this.

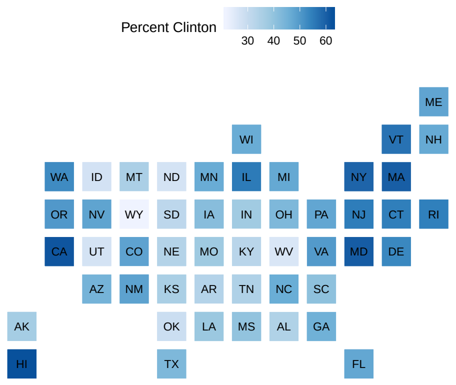


Figure 4: Statebins of the election results. DC is omited from the Clinton map to prevent the scale becoming unbalanced. Challenge: recreate a plot that looks like this.

Good Luck

The deadline for the assignment is **December 5th, 10am**!

References

Healy, K. (2018). *Data visualization: a practical introduction*. Princeton University Press.

Alex Hana's Github

Alex Hana's Github