3. Making sense of spatial data

Kimbal Marriott

3. Making sense of spatial data

Kimbal Marriott

Contents

1 Making sense of spatial data: Overview

This is the third module in the FIT5147 Data Exploration and Visualisation unit. In this module you will learn about common graphics for showing spatial data. That is, data that is naturally associated with a geographic location or region and for which you want to explore the data taking this association into account.

Aims of this module

After completing this module you will:

- have seen standard visualisations for showing spatial data and know when to use them;
- have the ability to choose appropriate map projections and an understanding of their limitations;
- have knowledge of common GIS tools and data formats;
- have first-hand experience with R and the graphics package **ggmap** and **leaflet** to construct map-based visualisations.

How to study for this module

In this module we draw on books, journal and conference articles as well as material in the public domain, including a few videos.

There is one associated assessment activity

- programming a spatial visualisation with R;

You should also continue your data exploration and visualisation project. By the end of the module you should have decided what you wish to communicate and started to design your interactive presentation visualisation.

2
How to design a data map

Here we will look at the various decisions and steps involved in designing a data map (or *thematic map* as they are usually called by cartographers). The first question is whether you should actually be using a data map. In short, you should use a data map if the data is associated with a geographic location or region and you wish to explore or show how this affects the other attributes. Just because data has a geographic location doesn't necessarily mean that you should show it on a data map. For instance if you are only interested in comparing sales figures for different stores in a company you would better to compare these using a bar chart. However if you are interested in seeing how the location of the store affects sales figures then you should use a data map. This might, for instance, show sales figures for the stores overlaid on a choropleth map of average income broken down by suburb.

Before you read further, take a look at some of the data maps Ben Wellington uses in his TED Talk [Making data mean more through storytelling](https://www.youtube.com/watch?v=6xsvGYIxJok) (https://www.youtube.com/watch?v=6xsvGYIxJok) (14min).

Data maps are one of the more difficult visualisations to design: it is easy to inadvertently create a misleading data map.

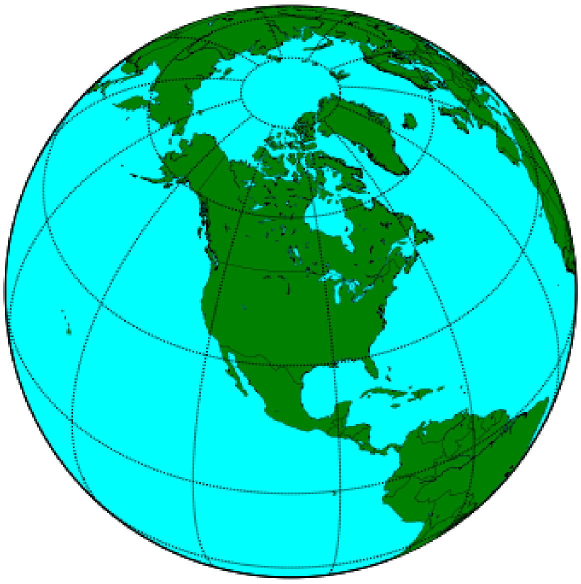
Scale and generalisation

Maps can be drawn at very different levels of scale. The scale of a map is often expressed as a ratio of map units to Earth units. This is called the *representative fraction*. For instance, a scale of 1:10,000 means that 1cm on the map represents 10,000cm = 100m on the earth's surface. Map scale can be shown using a representative fraction or a graphical bar scale or even given verbally as in "1 inch to the mile." Graphical bar scales are by far the most easily understood way of showing the scale of a map and should be included whenever the scale is relevant to understanding the data and the scale is relatively consistent over the map.

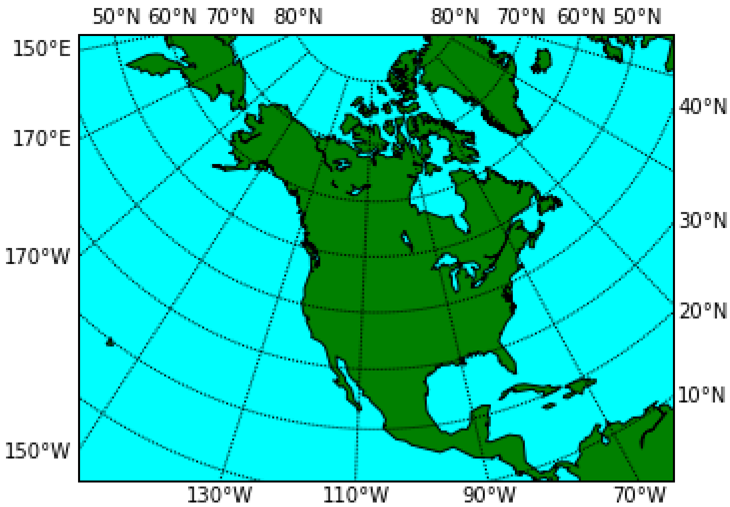
The scale of a map dictates how much detail can be shown. A *large scale* map shows a small area in detail, while a *small scale* map shows a large area in less detail. Spatial data sets often contain very detailed information and one part of map design is working out what is the appropriate level of detail to show in the map and then aggregating or generalising the geographic and non geographic data appropriately. For instance, imagine you wish to compare population density. On a map of the world you might show country boundaries, on a map of Australia you might show state boundaries and the capital cities.

Coordinate system

The Earth is roughly speaking a sphere though it bulges a bit at the equator because of the centrifugal forces generated by rotation. Location on the earth's surface is given by *latitude* and *longitude*. The location of Melbourne is given by 37°48′49″S 144°57′47″E.



Latitude lines go from east to west and are parallel to the equator. They run from 90^{o}N which is the North Pole to 0^{o} which is the equator down to 90^{o}S which is the South Pole. Longitude lines run at right angles to the latitude lines and are circles running north to south through the poles. Longitude are measured east or west from Greenwich in the UK. They range from 180^{o}W to 0 which is Greenwich to 180^{o}E (which is of course equal to 180^{o}W). Probably like me you can't remember which is which: lines of longitude are all "long" while lines of latitude vary in size. Lines of latitude are often called parallels as they do not intersect while lines of longitude are called meridians and the meridian through Greenwich is called the Prime Meridian.



Map projections

One of the great difficulties facing map-makers is how to faithfully show the surface of the Earth (a flattened sphere) on a flat piece of paper. A *map projection* is the mapping between points on the surface of the Earth and the position on the two-dimensional map. You might like your map projection to avoid distorting angles, distances, area and directions. Unfortunately this is impossible: there is no 2D projection that can do this.

For maps only showing a small part of the Earth's surface the distortions are small but for maps showing continents or the entire world the distortions are considerable. As a result dozens of different map projections have been invented. There is no single best projection: the appropriate choice depends upon the task, area covered by the map and the kind of data map. Thus, choosing the right projection is am important part of designing a data map.

Projections can be classified depending upon which property they preserve

- *Equal area (or equivalent) projections* preserve area.
- *Conformal projections* preserve angles locally,
- *Equidistant projections* preserve distance from a particular location
- *Azimuthal projections* preserve directions from a particular location, while
- *Compromise projections* ensure that area and angle distortion is "not to bad."

The most commonly used map projection is that of Mercator. This was invented by Gerard Mercator in 1569. His map was designed to help European sailors who urgently needed nautical charts that allowed them to easily navigate between very distant locations. His map uses a rectangular grid of meridians and parallels. For the meridians to be straight lines the horizontal scale must increase as the latitude increases/decreases moving away from the Equator since the actual distance between meridians decreases until they meet at the North and South poles.

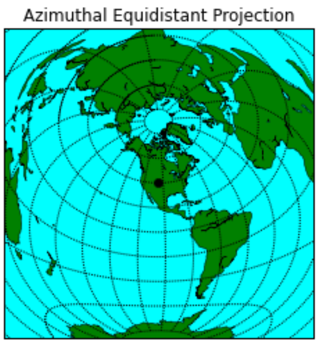
The clever part of Mercator's projection is to smoothly increase the vertical scale as the latitude increases so that the vertical scale always remains the same as the horizontal scale. As a result the distance between the parallels increases as you move away from the Equator even though they are equally spaced on the actual Earth itself. By linking the vertical scale with the horizontal scale Mercator ensured that regions drawn on the map locally retain their shape since the scale around any point on the map was consistent in all directions. Even better, Mercator's projection ensured that a straight line drawn between two points on the map gives the correct bearing in which to travel between the points.

Of course this comes at a price: the scale is not consistent across the map and regions at high latitude are disproportionately large: Furthermore it is not even possible to show the North or South Pole. Nonetheless, the Mercator projection has become *the* standard map projection, the one that is used on most maps including those of Google and the one that most of us learn our geography from. It is because of this that many of us still think that Greenland is as big as Africa.

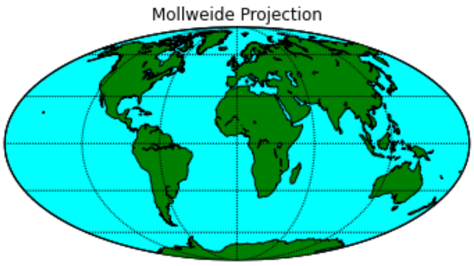
However, the use of the Mercator projection should not be automatic. While it is familiar, unless you need to navigate around the world it is almost certainly not the best choice of map projection. In 1989 the Committee on Map Projections of the American Cartographic Association released a resolution strongly advocating against the use of rectangular world maps including those based on Mercator's projection because of the serious, erroneous conceptions that they create.

Table 3.2 from *Cartography: Thematic Map Design* provides a guide to which projection to choose for different purposes and regions. A very reasonable choice for data maps of the world is to use either the *Winkel Tripel Projection* or the *Robinson Projection* as both are pleasing looking compromise projections that minimises area and angular distortion. However, if you are creating a data map of the world in which area is important in interpreting the data such as when using a dot map to show total population and density then the *Mollweide Projection* is a better choice because it is an equal area projection. For showing continent sized areas, a good choice is the *Lambert Azimuthal Equal Area Projection* appropriately centered on the region. If you are creating a flow map for flows from a single origin and the direction and distance of flow is important, then the ideal choice is an *Azimuthal Projection* centered on the origin appropriately scaled so that it preserves distance as well as direction from origin to destination.

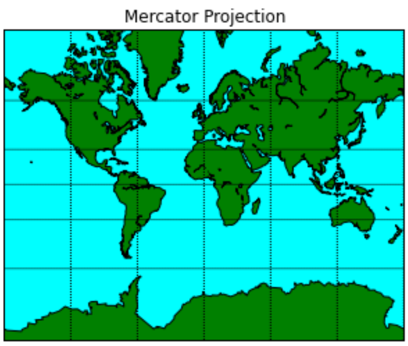
It is worth emphasising that when choosing the map projection you can also choose where to center the map and the point or line(s) for which distortion is minimised: obviously you should choose these so as to minimise distortion of the most importance areas. For smaller regions the choice of map projection is not as important as the amount of distortion is less.



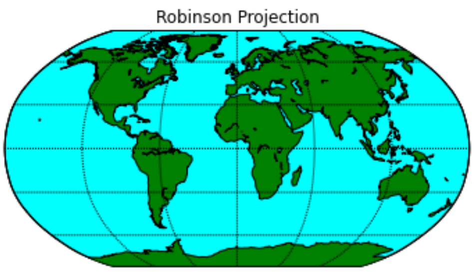
(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/AzimuthalEquidistant.png)



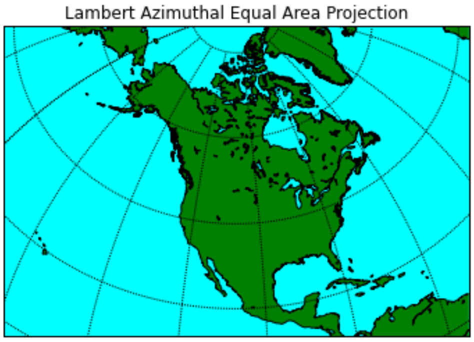
(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/Mollweide.png)



(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/Mercator2.png)



(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/Robinson.png)



(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/LambertAzimuthalEqualArea.png)

All based on: http://matplotlib.org/basemap/users/mapsetup.html (Python)

Given the problems with projection it is sometimes worth thinking about another approach: actually showing the Earth as a 3D globe. The disadvantage of this is the difficulty of navigating in 3D space and that curvature means it is impossible to see more than half the globe at once and that depth introduces difficulties in accurately comparing distances and errors. However it does have the advantage that there is no built in bias.

What you favourite map projection says about you here (http://xkcd.com/977/)

Choice of data map

The next choice when creating a data map is which of the different kinds of data maps to use.

Choropleth and prism

Choropleth mappings show categorical and ranked (both ordinal and quantitative) data associated with regions that have fixed boundaries. These regions are typically political or administrative boundaries. The data value is usually represented using colour though sometimes a pattern is used and regions on the map are filled with the colour or pattern corresponding to their data value.

A prism map is a variant of a choropleth map in which height is used to show the data value. Prism maps are well suited to comparing quantitative data.

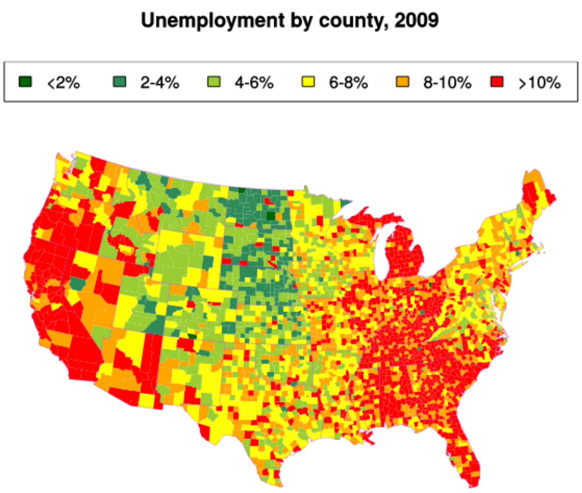
If using grey-scale it is conventional to use darker colours for higher values and lighter colours for lower values.

Choropleth maps can either plot unclassed data, in which case a continuous scale is used, e.g. from white to black, or data segregated into classes in which case distinct colours are associated with each class. It is imperative that a legend is associated with the map, so as to allow the colour encoding to be understood.

If the area of the regions varies this can affect how the data is understood. Thus for example, if you show total population for countries, then the extent of large, populous countries such as Russia makes them visually dominate smaller, populous countries like Indonesia. Rather than showing total population it is better to show population density if you are using a choropleth map. In general it is better to show densities rather than totals with choropleth maps.
One drawback of choropleth maps is that the data value is shown uniformly throughout the enumeration

area but in many cases the real distribution such as population density is not uniform across the region. *Dasymetric* maps instead use regions, called zones, that have more uniform value. These are computed from the data or inferred from other data sources such as for instance housing density. Once the zone boundaries are computed their data is represented in the same way as in a choropleth map.

The choice of colour in a choropleth map is important. We will look at this in more detail when we study the human visual system. (https://www.alexandriarepository.org/module/placeholder-human-visual-system/).



(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/ChoroplethColour.png)

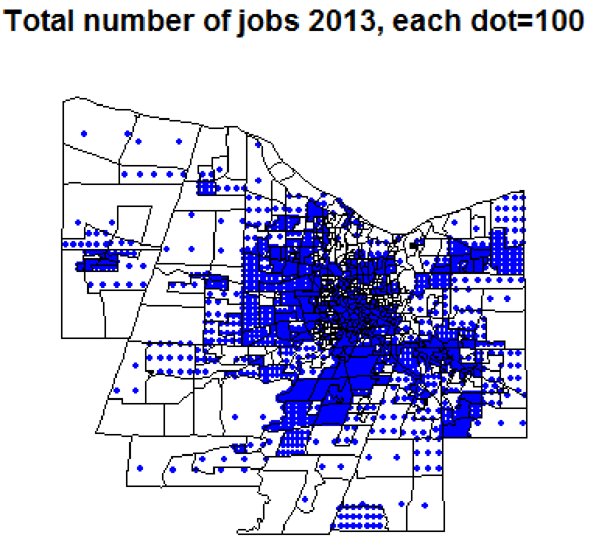
Based on: https://cran.r-project.org/web/packages/maps/maps.pdf

Dot density

Dot density mapping shows quantitative data associated with regions that have fixed boundaries. Dots or some other symbol are drawn on the region in a semi-random distribution. The number of dots is proportional to the data value. It is important to provide a legend showing representative densities, typically low, medium and high.

The disadvantage of dot density mapping is that the reader may see patterns in the semi-random distribution that do not actually exist and that reader perception of differences in dot densities is non-linear and it is difficult to see small differences. The advantages are that it can reveal overall patterns and it is suited to multivariate visualisation by using different colours or symbols for different values.

Dot placement should be sensitive to land use. For instance when creating a dot map of the population of councils in Melbourne it is a good idea to not place dots on parklands, rivers, lakes or the sea even though these lie within council boundaries.



(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/DotDensity.png)

Based on: https://github.com/mikeasilva/dot-density-map

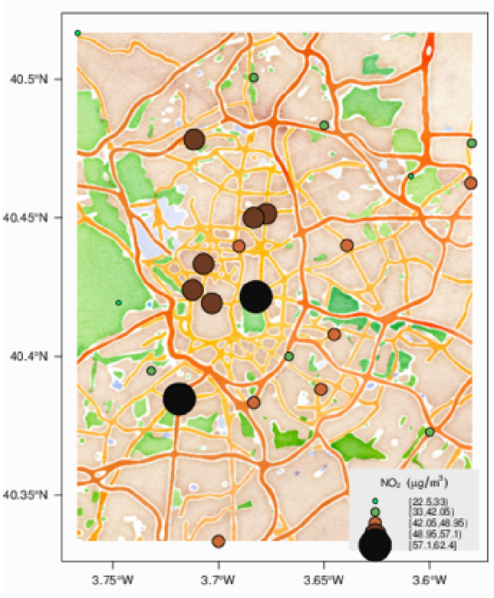
Proportional symbol

In this kind of map a symbol (such as a circle, triangle, square or pictorial icon) is used to show ordered data associated with either a region or a particular location. The size of the symbol is proportional to the data value. One kind of ordered data that it is not suitable for is interval data, that is quantitative data without a natural 0. Thus temperature should not be shown with this kind of map while population can be. This is because there is a visual implication that absence of the symbol means 0.

Care should be taken in the choice of symbol as it is for instance, harder to compare the area of two circles than that of two squares while comparing the area of two stylised airplanes is very difficult. It is important to provide a legend showing representative sized symbols, typically 3-5 values.

Most proportional symbols vary the symbol size continuously. However, *range grading* in which attribute data is divided into groups is also used. This has the advantage that the symbols for each group are clearly distinguishable. Range grading is the same process as class segregation in choropleth mapping.

More generally, we can associate more complex statistical graphics such as pie charts, bar charts, time series, etc with regions or locations on a map. This is one way to handle multivariate spatial data but great care needs to be used to make sure that these graphics are eligible and comparable.



(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/Symbols.png)

Proportional Symbol (air pollution) based on: http://oscarperpinan.github.io/spacetime-vis/spatial.html

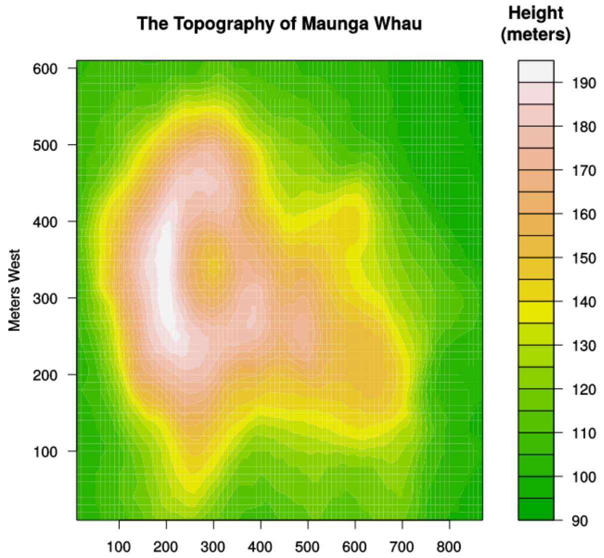
Contour (isarithmic) & three-dimensional maps

A useful metaphor for visualising quantitative data that varies continuously across space is to map it to a kind of conceptual height or elevation for each point on the map. The resulting 3D map is then visualised using standard techniques for showing elevation like contours, wire-frames, surfaces or shaded relief.

Properly speaking the contours are called *isolines* and the resulting maps are called *isarithmic* maps. When using contours, it is helpful to add colour tinting to clearly show the regions between each pair of contours.

Personally I like 3D maps and with the rise of cheap 3D visualisation I think their use will increase.

The main thing to be aware of is that they only make sense for data that varies continuously, not data that has abrupt changes or discontinuities. Thus they are ideal for showing temperature and possibly for showing crop yield per acre but not so good for showing the distribution of coal mines.



(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/Contour1.png)

Based on https://stat.ethz.ch/R-manual/R-devel/library/graphics/html/filled.contour.html

Cartograms

Cartograms are also called value-by-area maps. They are for mapping quantitative data with a natural zero, such as population or average income, to regions.

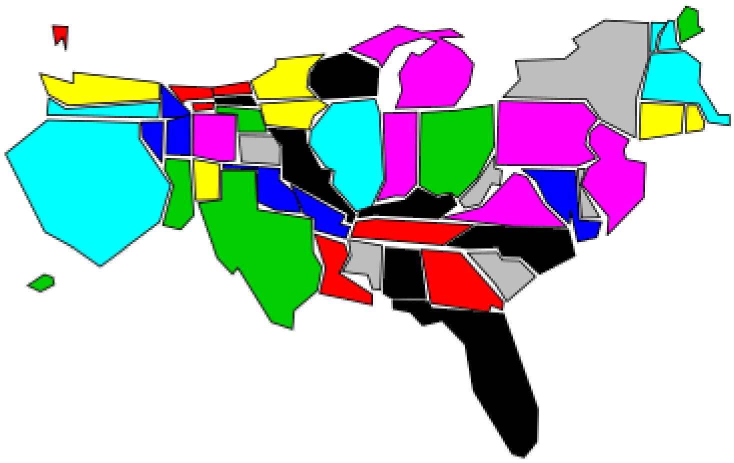
In a cartogram a spatial transformation is applied to the region boundaries so that the region area is proportional to the data value. Two forms of cartogram exist: *contiguous* and *non-contiguous.*

In a contiguous cartogram the regions remain connected but are distorted to change their area. The transformation attempts to preserves shape as much as possible so that the regions are still recognisable.

In a non-contiguous cartogram the map is "exploded" so that there is sufficient room between the regions for them to grow in size. In this kind of map the regions preserve their shape but how they fit together is more difficult to see. In a more abstract variant the regions are replaced by a simple geometric shape such as a circle.

Cartograms require the reader to be familiar with the underlying geography. Because of the irregular shape of the regions it is difficult to compare their area so they can be difficult to understand. Thus I would not recommend their use in data exploration.

However, they can powerfully communicate information such as global health or wealth inequality to stakeholders. For lots of interesting examples take a look at [Worldmapper](http://www.worldmapper.org/index.html) (http://www.worldmapper.org/index.html).



United States mainland Cartogram, roughly proportional to population (California is full).
Based on: https://cran.r-project.org/web/packages/maps/maps.pdf

Flow maps

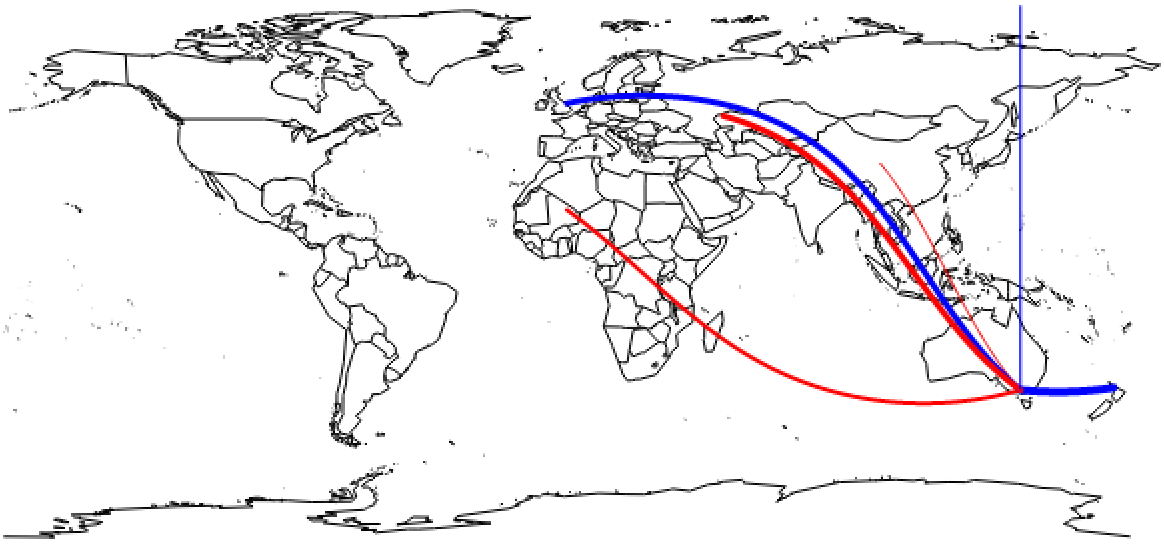
Discrete flow maps show the movement of commodities such as goods or people between places. They are also called *origin-destination* (OD) *maps*.

Most commonly lines connect the origin of the flow with the destination. Arrows indicate the direction of flow while line width or sometimes colour shows the magnitude. Lines are typically bundled together and curved to reduce overlap and cluttering. Proportional symbols may be used to show the magnitude of total flow out of origins and total flow into destinations. It is important to provide a legend showing representative sized symbols and flows, typically 3-5 values.

Such maps work well when there are only a few origins or destinations but with many origins and destinations the lines cross and clutter makes it difficult to understand the underlying data. Interactive exploration or aggregation are necessary in this case.

Flow maps can also be used to show the magnitude and direction of continuous flows such as wind patterns or magnetic patterns.

Often we wish to understand how flow changes over time. Animation is often used for this. Another approach is to use a *space-time cube*, a kind of 3D discrete flow map where time is the third dimension.



Email flow to (red) and from (blue) my secret lair (in Melbourne), including one to Santa and some spam from Africa, China & somewhere in Russia. Line thickness represents quantity. Created with R using library(maps) & library(geosphere). Note the curved 'great circle' lines.

Choice of classes

One issue when using a classed choropleth map to show continuous data or range graded proportional symbols is the choice of data classes. The more classes the more complex and sometimes confusing the visualisation becomes but fewer classes can hide or misrepresent the underlying data. When exploring data I would tend to use lots of classes while for presenting results the number of classes should be smaller and designed to show the result you wish to communicate.

A typical number of classes is 4-5 though more classes may be used for large amounts of data. Sturges suggests that the number of classes, *C*, should be $C = 1 + 3.3 \times \log(n)$ where *n* is the number of observations.

Once you have decided the number of classes then you have to decide how to partition the observations into these classes: that is the minimum and maximum value for each class. The class intervals should not overlap and should contain all observations. This is an example of a one dimensional clustering problem. Many techniques are used but among the most common are

- *Equal Interval*: Simply divide the data range into *C* equally sized intervals.
- *Equal Frequency*: Compute class intervals so that they have the same number of observations. Thus the intervals correspond to quantiles. If there are 4 classes then the classes correspond to the quartiles.
- *Jenks Optimisation*: This chooses the classes so as to minimise the variance within each class while maximising the variance between the classes. An iterative algorithm is used to do this.

All of these are reasonable approaches. For exploration Jenks Optimisation probably makes the most sense but it can be difficult to explain to non data scientists.

Aggregation and clustering

A key part of data map design is to choose the level of detail in which you wish to show the spatial data. Too much detail can overwhelm the viewer while too little detail may mean that the reader is lulled into thinking that the data is more homogeneous than it really is. In interactive visualisations the level of detail can be controlled by the user, either implicitly through zooming or explicitly by a control.

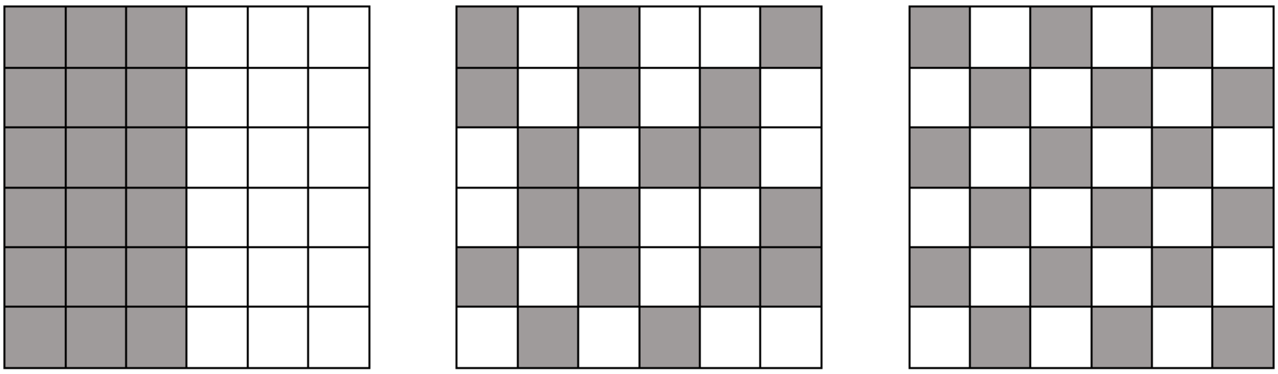
Spatial aggregation-combining data from adjacent regions to form a bigger region can be quite misleading if not done well. This is because the choice of region boundaries can greatly affect the results of aggregation. As an example the boundaries of electoral districts can change the result of elections and gerrymandering is one way in which politicians can ensure they win an election even though most people do not vote for them.

Show "Gerrymander.tiff"
(https://docs.google.com/viewer?url=https%3A%2F%2Fwww.alexandriarepository.org%2Fwp-content%2Fuploads%2F20170222150254%2FGerrymander.tiff&hl=en_GB&embedded=true)
Download (TIFF, 17KB) (https://www.alexandriarepository.org/wp-content/uploads/20170222150254/Gerrymander.tiff)

If cells are grouped horizontally rather than vertically aggregation gives a very different result.

Spatial autocorrelation

One of the main reasons for visualising data on a map is to see whether data values are correlated with spatial location (called *spatial autocorrelation*). Spatial autocorrelation can also be tested using statistics like the Moran coefficient (https://en.wikipedia.org/wiki/Moran%27s_I) (Moran I) or Geary's C (https://en.wikipedia.org/wiki/Geary%27s_C). When exploring data it is wise to confirm the the correlation identified visually is statistically meaningful.



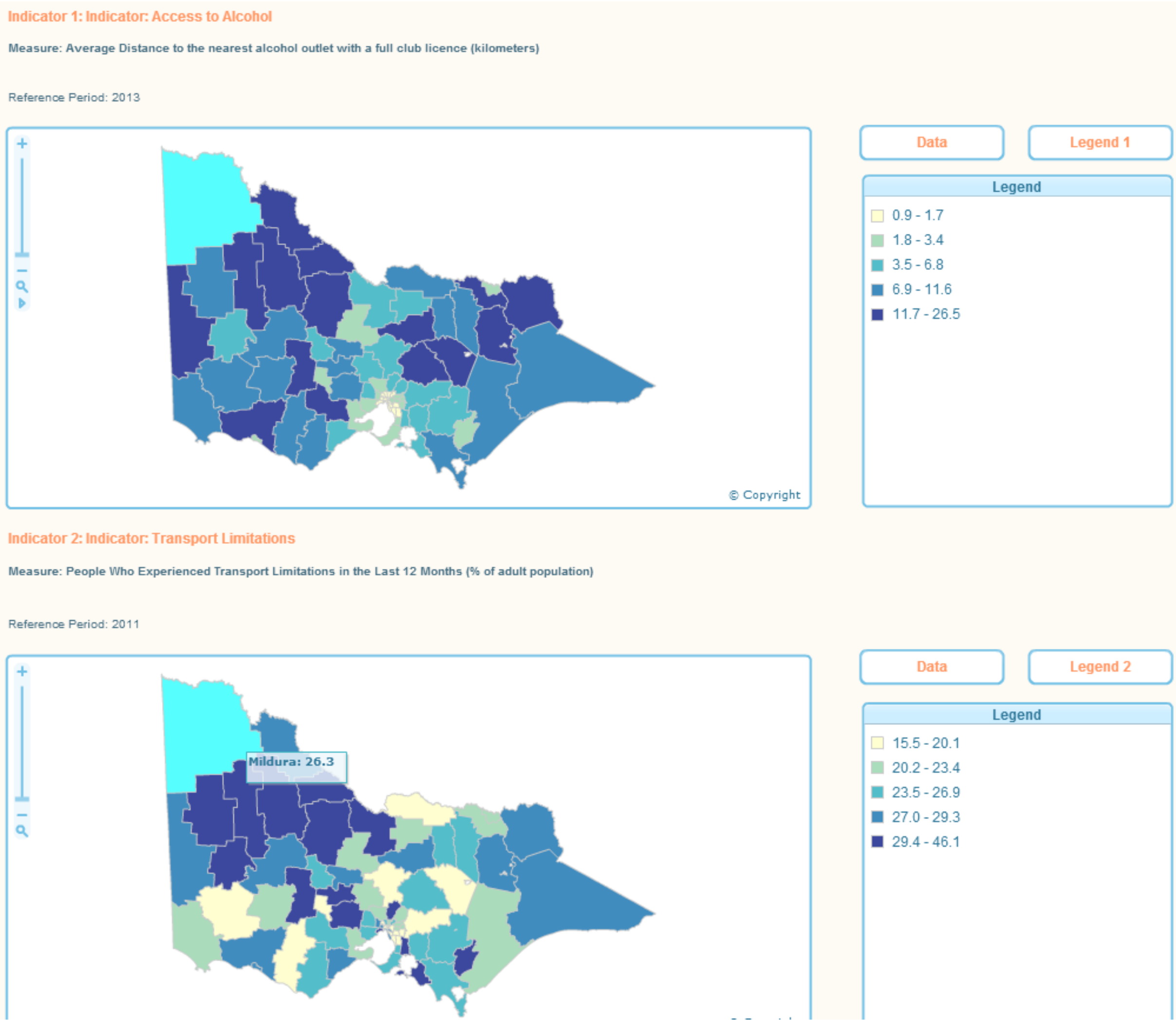
Left: positive spatial autocorrelation; middle: random distribution so no spatial autocorrelation; right: negative spatial autocorrelation.

Multivariate visualisation

We have focused on showing the spatial distribution of a single variable or attribute. However, in most real world applications data has many attributes and it is the interconnection between these different attributes that the data scientist is most interested in exploring. Unfortunately this is difficult as showing the data on a data map means that two visual variables (x position and y position) are already used up and other variables must be used to show different attributes.

Some approaches are to use two colours or a mix of colour and texture on a choropleth maps and more complex glyphs on proportional symbol maps. However the most common approach is to show multiple data maps side-by-side with a data map for each attribute or to overlay two data maps. Interaction allows filtering and brushing to show connection between the different attributes or to toggle between the overlays.

(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/BrushingCIV.png)



(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/BrushingCIV1.png)

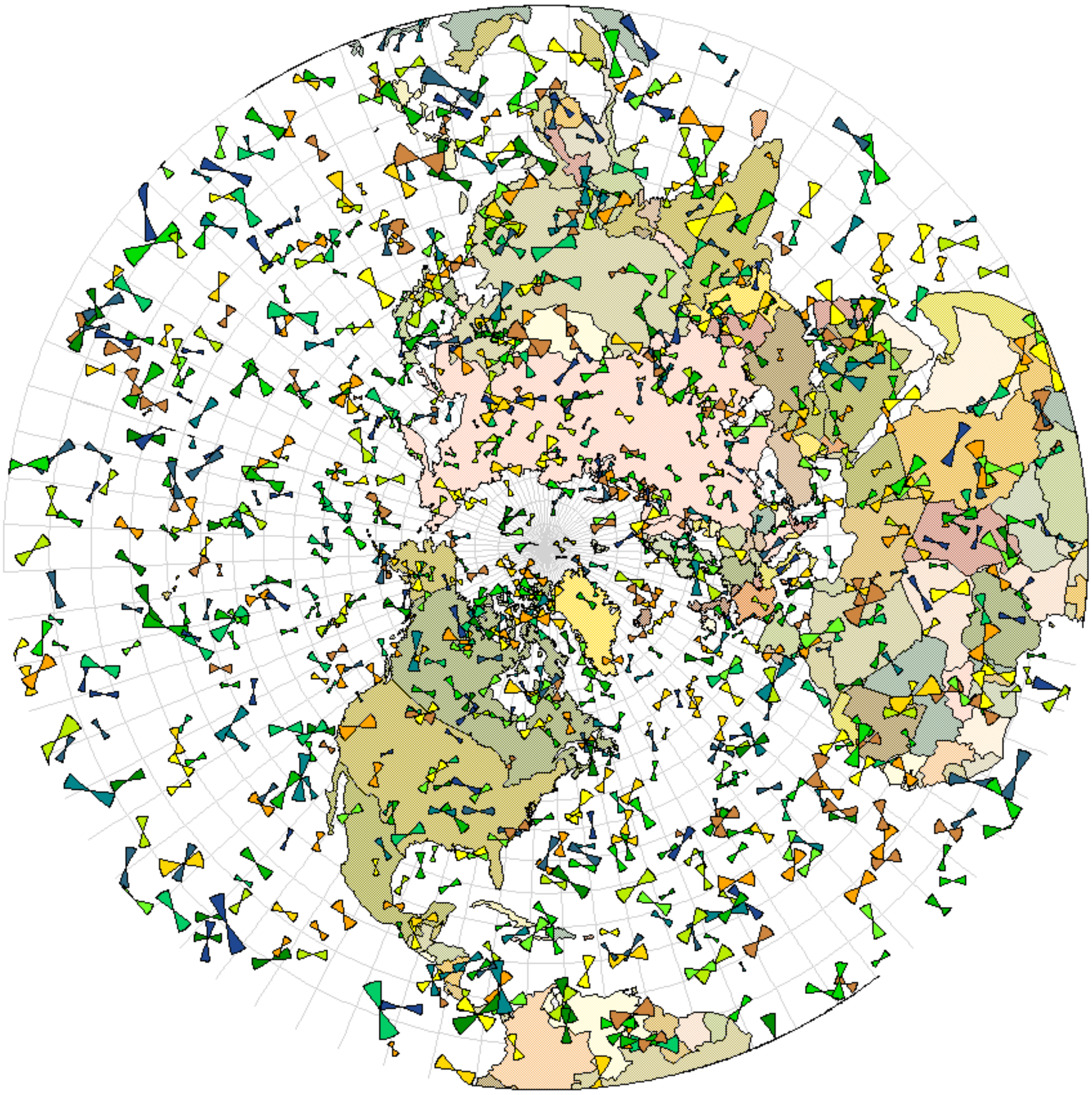
Multivariate data with brushing, Mildura is highlighted in both maps (but not both data points…).
The maps show Access to Alcohol and Transport
From: http://www.communityindicators.net.au/files/DoubleMap/atlas.html

Uncertainty

One of the disadvantages of visualisations and maps in particular is that they can mask uncertainty and errors in the data and encourage the viewer to have more trust in the patterns than it warrants. Good data map design should ensure that the visualisation does not install false confidence in the data or patterns.

Uncertainty arises in a number of ways: sampling uncertainty or measurement errors in attribute values and positional accuracy are the most common.

There are two main ways of showing uncertainty. *Intrinsic approaches* tightly couple the visualisation of uncertainty with the attribute being visualised. Specialised visual variables including crispness, resolution and transparency are commonly used in intrinsic approaches. On the other hand extrinsic approaches decouple visualisation of uncertainty from visualisation of the attribute and its spatial location. *Extrinsic approaches* include error bars or a separate data map showing confidence levels.



(https://www.alexandriarepository.org/wp-content/uploads/20150929031513/bowties.png)

B*owtie plots* can be used on a map to show uncertainty in the observed direction (called azimuthal data) to some point of interest. The centre of the bowtie is the location of the observation, the orientation of the bowtie is the direction while the width gives the uncertainty. This is similar to shaded regression lines. Based on: http://gis.stackexchange.com/questions/31294/how-to-visualize-azimuthal-data-with-uncertainties

Summary

In this section we have discussed the various design decisions that need to be made when creating a data map: choice of scale, choice of map projection, choice of data map type and how to group continuous data into classes when this is required. Without considering these choices carefully it is easy to create data maps that are uninformative or even worse, misleading.

On the other hand a well designed data map is one of the best ways for exploring spatial data that we have and can be an extremely powerful way of communicating the resulting insights.

One of the most highly regarded data visualisations of all time is a data map, by Charles Minard and shows Napoleon's march on Moscow during the 1812-13 Russian campaign (here (https://en.wikipedia.org/wiki/Charles_Joseph_Minard#/media/File:Minard.png)). It uses a flow map style graphic to show the movement of Napoleon's army during the march to Moscow and subsequent retreat. The thickness of the line is proportional to the number of troops. It is a sophisticated multivariate graphic showing the number of soldiers; distance traveled; temperature; troop position and relevant geographic features; direction of travel; and time taken.

FURTHER READING

This material was based upon

Cartography: Thematic Map Design (6th Edition). Borden Dent, Jeff Torguson, Thomas Hodler. McGraw-Hill. 2009.

Thematic cartography and geovisualization (3rd Edition). Terry Slocum, Robert McMaster, Fritz Kessler, Hugh Howard. Prentice hall, 2009.

Both are good introductions to the design of effective data maps.

And Cartographic Map Projections (http://www.progonos.com/furuti/MapProj/Normal/TOC/cartTOC.html) is a good resource about projections.

3
Overview of tools for creating data maps

There are now a wide range of tools that can be used to create data maps. Business intelligence tools like Tableau provide basic data map functionality while R and Python both have packages for creating data maps. Those in R include **ggmap** and **Leaflet**. However if you need to do a lot of spatial data analysis then you should think about using specialised GIS software.

GIS

Geographic Information Systems (GIS) date from the late 60s. A modern GIS can be used to capture, store, transform, analyse and visualise spatial or geographical data. Data maps are made by creating a different layers for different kind of data objects, such as towns, borders or rainfall. Elements in the layers have a spatial and sometimes temporal coordinate and the layers are overlaid to form the data map. GIS typically provide good spatial analysis tools and allow smooth transition between different levels of detail.

Spatial data is stored in a GIS system in two ways

- As *raster* or pixel based data. Arial photos, satellite images, and elevation data are typically stored as raster data.
- Geographic shapes are typically stored as *vector* data. Points are used to store location of zero-dimensional features like a mountain peak, lines or poly-lines to store the position of one-dimensional features like roads or rivers, and polygons to store the position and shape of two-dimensional features like lakes or countries.

GIS tools & resources

- Esri ArcGIS (http://www.esri.com) software is the industry standard in commercial GIS systems. It allows users to combine a wide variety of data sources to create data maps, comes with its own cartographic data and can perform spatial analysis.
- MapBox (https://www.mapbox.com), CartoDB (https://cartodb.com), MapInfo (http://www.pitneybowes.com/au/location-intelligence/geographic-information-system.html), are other commercial GIS tools. The first two currently offer limited functionality for free.
- QGIS (http://www.qgis.org/) is an open source GIS tool.
- Google provides Google Fusion Tables, Google Charts and APIs to Google Maps and Google Earth.
- National Map (http://nationalmap.gov.au) is a great online GIS tool for quickly viewing Australian government data
- Open Street Map (https://www.openstreetmap.org) (OSM) is a great online open source resource for maps.

GIS data formats

Unfortunately a wide variety of different formats are used by GIS software to exchange spatial data. JPEG2000 (https://en.wikipedia.org/wiki/JPEG_2000) or GeoTIFF (https://en.wikipedia.org/wiki/GeoTIFF) are often used for GIS raster data. Four common formats for GIS vector data exchange are

- Keyhole Markup Language (https://en.wikipedia.org/wiki/Keyhole_Markup_Language) (KML) - XML based open standard that underlies Google Maps
- Geography Markup Language (https://en.wikipedia.org/wiki/Geography_Markup_Language) (GML) - XML based

open standard developed by the Open Geospatial Consortium that also includes raster data.
- Esri shapefile (https://en.wikipedia.org/wiki/Shapefile) - widely used format developed by Esri
- GeoJSON (https://en.wikipedia.org/wiki/GeoJSON) - JSON format used by many open source GIS packages

There are tools to convert between these different formats. Vector data may also be specified using SVG (https://en.wikipedia.org/wiki/Scalable_Vector_Graphics) the W3C Scalable Vector Graphics standard.

4
Activity: Exploring & Visualising Data with Google Fusion Tables

Google Fusion Tables

Creating data maps doesn't need to be hard work. There are a number of easy to use tools for making them.
One of these is a tool provided by Google, called Fusion Tables, that allows you to easily overlay Google Maps with additional data. In the following activities we are going to demonstrate the use of Google Fusion Tables for geographic data visualisation and fusion, i.e. joining databases together (and laying them over a map). You need to have your own gmail account, or choose to 'opt-in' to this Google service using your Monash gmail account (see [GDDS orientation](http://moodle.vle.monash.edu/mod/choice/view.php?id=2752879) (http://moodle.vle.monash.edu/mod/choice/view.php?id=2752879), Google+ opt-in). Given that this is a Google tool it's recommended that you use a Google browser (Chrome).

A. Data on top of a (Google) map

Step 1

Download the dataset (butterflies!)
https://sites.google.com/site/fusiontablestalks/astraptes_fulgerator_complex_sample_data.csv

And inspect:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	herbivore species	voucher	URL adult	URL cp lat	sex	host plant species	host plant fam	year	date eclosic	elevation	wingspan	primary eco	latitude	longitude
2	Astraptes SENNOV	05-SRNP-59407	http://janzen	http://jan	female	Senna hayesiana	Fabaceae	2005	9/11/2005	560	60	dry forest	10.75876	-85.3754
3	Astraptes INGCUP	07-SRNP-55016	http://janzen	http://janzen.sas.up		Inga sapindoides	Fabaceae	2007	1/29/07	305	59	dry forest	10.77175	-85.434
4	Astraptes FABOV	80-SRNP-202	http://janzen	http://jan	male	Senna hayesiana	Fabaceae	1980	7/28/80	290	46	dry forest	10.85145	-85.608

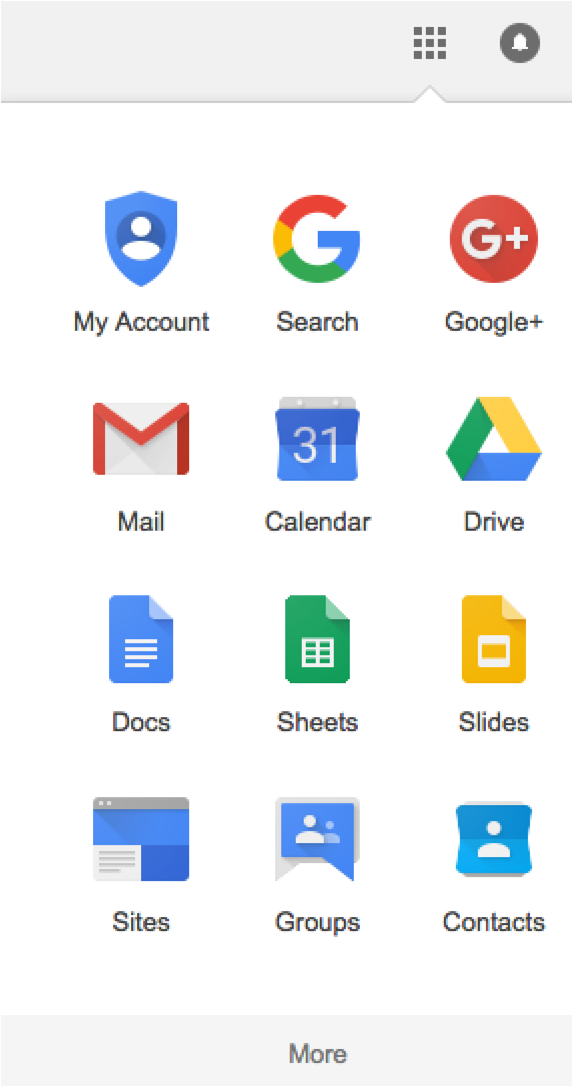
The first column is the butterfly name, they are all types of 'Astrapes' (https://en.wikipedia.org/wiki/Astraptes).
Note the latitude and longitude, far right, almost identical. Note also the difference in date format (highlighted, see if Fusion Tables can handle this).

What's the largest butterfly, the smallest? What is 'eclosion'?

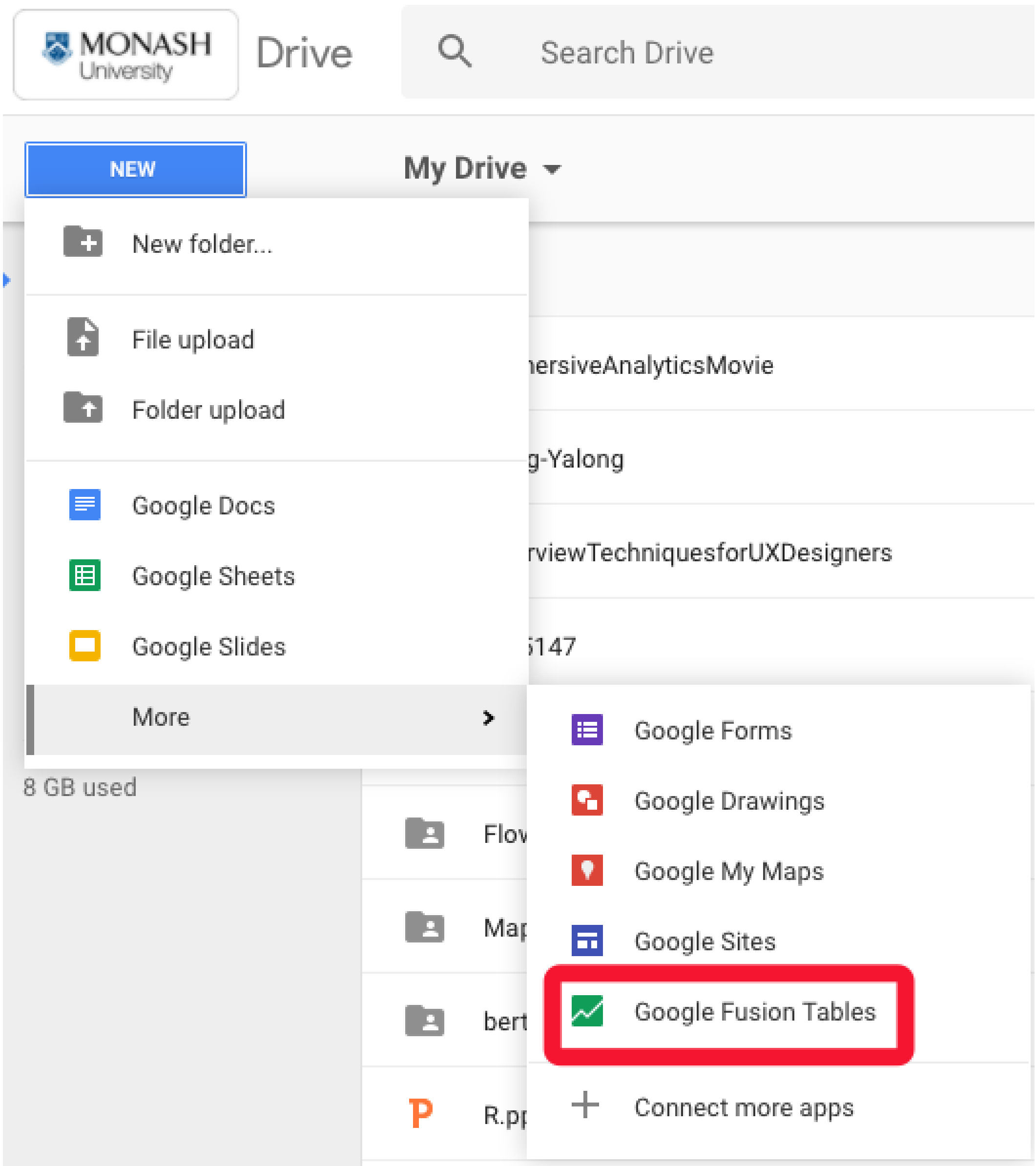
Step 2

Open your gmail, go to your google drive by click the right top of the page.

And Click "Drive". (you can of course go to your Google Drive directly.)

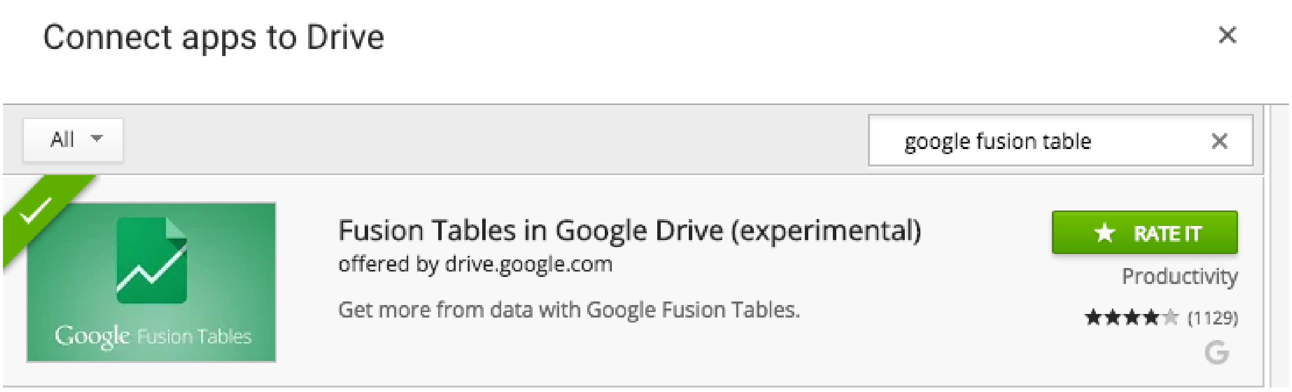


And create a "Google Fusion Table" (New > More > Google Fusion Tables:):

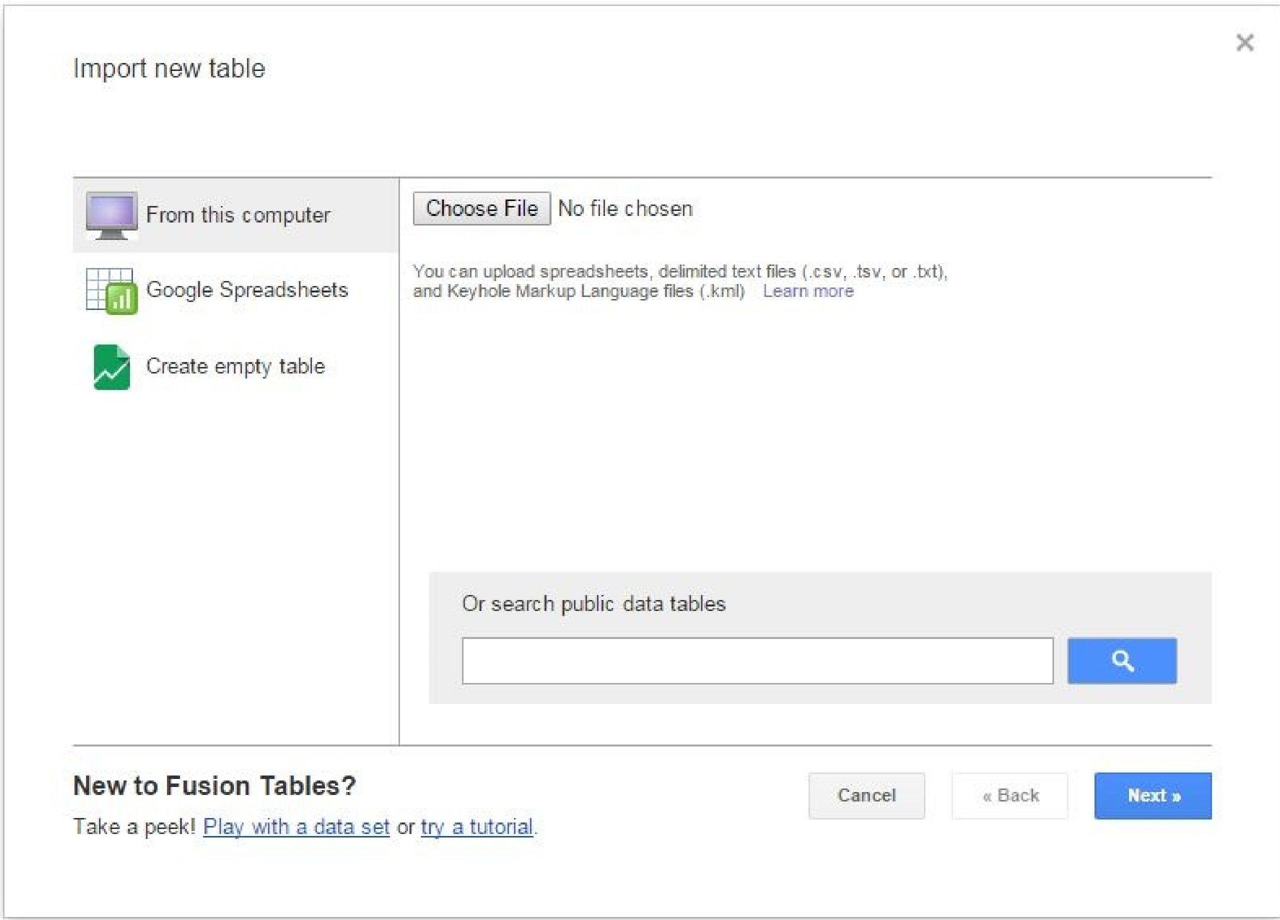


If you cannot see "Google Fusion Table", click "Connect more apps" then search for it, add it to your google service.

After that, you can then do the above action.

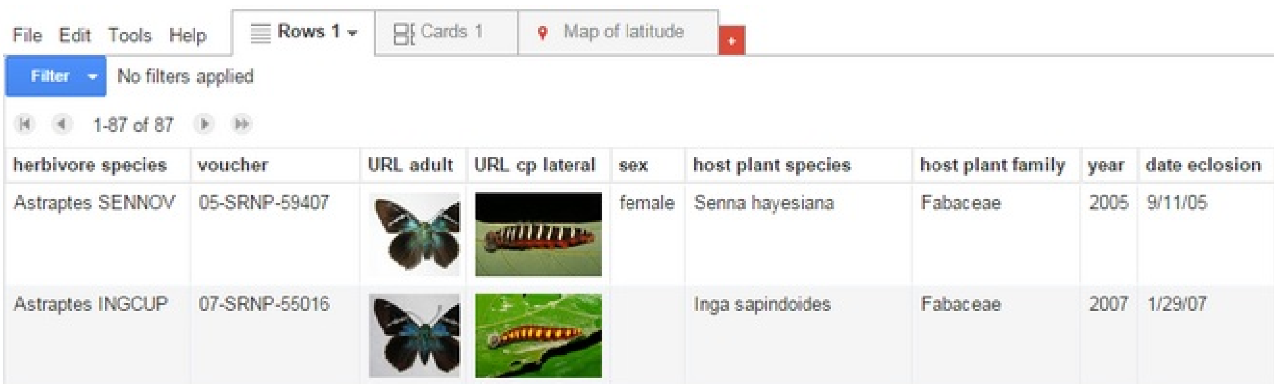


Then you will see a new page with:



Step 3. Load data

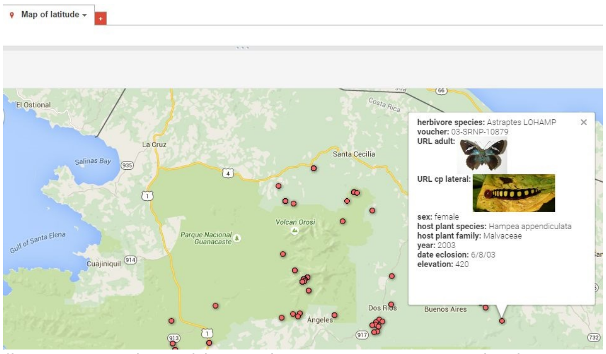
1. In the Import new table dialog box, click 'Choose File', find the butterflies data ("astraptes_fulgerator_complex_sample_data.csv") file you downloaded, and 'Open' then 'Next'
2. Preview the data but don't change anything then 'Next'.
3. Give your table a name and 'Finish' to see:



Note the missing data in the second row - how will it be treated?

Step 4

Fusion Tables auto-detects location data in a table and displays a tab called "Map of..."
In this case, the Map tab is titled "Map of latitude." Have a look.
What country is this?



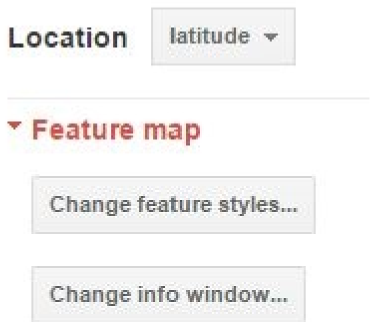
(https://www.alexandriarepository.org/wp-content/uploads/20151006083157/GFT5.jpg)

We can see scattered red dots, select one to see that it represents one data record.

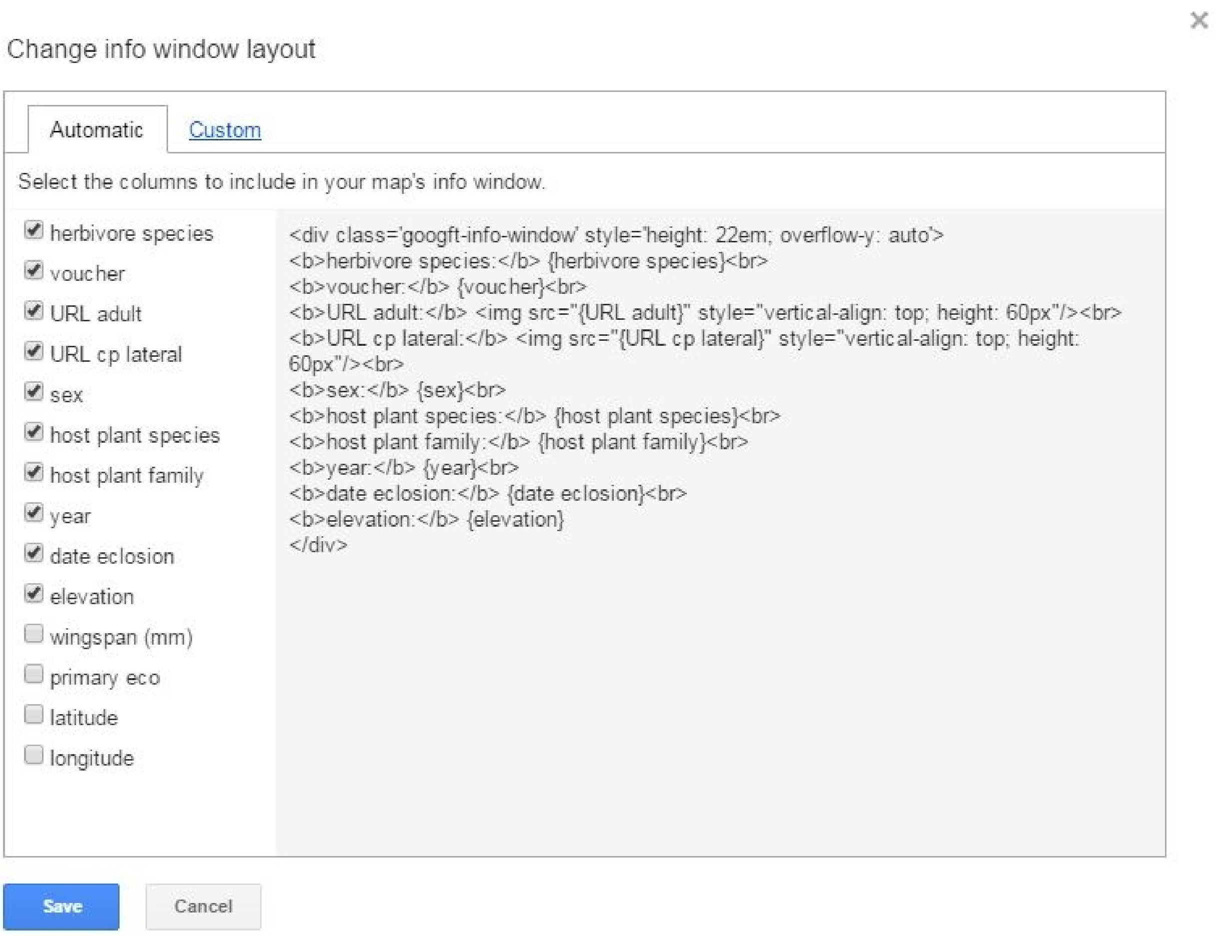
Step 5. Customize the map

The default info window template automatically uses the first ten columns for this table, but you can customize which data appears and how it is displayed.

Look at the left part of the page, click the "Change info window..." button.



to see:



You can experiment by checking column checkboxes to add or remove information from the Automatic info window template, notice the corresponding HTML changes. Then 'Save', then view.

To customize the overall style and content of the info window template, click the "Custom" tab (shown above). Replace the existing HTML with the following:

```
<div class="googft-info-window" style="font-family: sans-serif; width: 330px;
height: 20em; overflow-y: auto;">
    <img src="{URL cp lateral}" style="width: 150px; vertical-align: top;
margin-right: .5em" />
    <img src="{URL adult}" style="width: 150px; vertical-align: top" />
    <h2 style="color: brown">{herbivore species}</h2>
    Caterpillar found feeding on host plant
    <em>{host plant species}</em>,
    {host plant family} in the {primary eco},{year}.

    See specimen record {voucher} at
    <a href="http://janzen.sas.upenn.edu/">
        Caterpillars, pupae, butterflies and moths of the A.C.G.
    </a>
</div>
```

To see:

Step 6

Customise to suit your own tastes, e.g. add latitude & longitude, size, or even make the image proportional to the size of the butterfly...

Step 7

You can also use the 'Cards' menu to filter.

Try gender, how many null values are there?

How many butterflies have a wingspan of 45-50mm?

How many butterflies were found above 1,000m?

Based on https://support.google.com/fusiontables/answer/2527132?hl=en&topic=2573107&ctx=topic

Can you create non-map data visualisations with Google Fusion Tables?

B.Fusing tables and custom maps

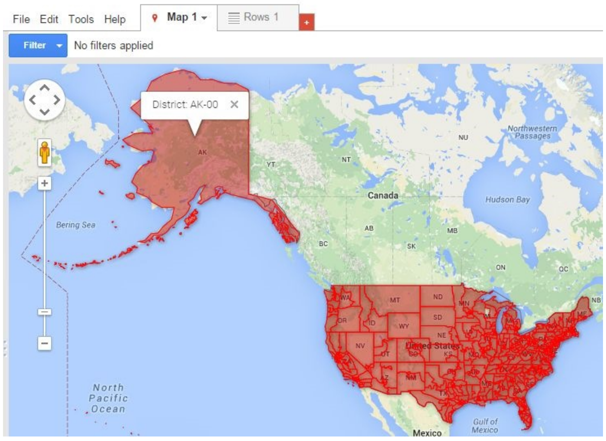
This exercise shows you an example of creating a highly-customizable choropleth map using two data sources. One dataset has some statistics about Congressional Districts (USA), the other table has their boundaries (which we can show on a map).

1. Start by combining the two public tables into a new, merged table.
2. Then select the Map visualization, and
3. Customize your map with a colour display based on the data (Choropleth Map).

Step 1

Combine data and boundary tables. Open the link of US Congressional District Boundaries, 110th

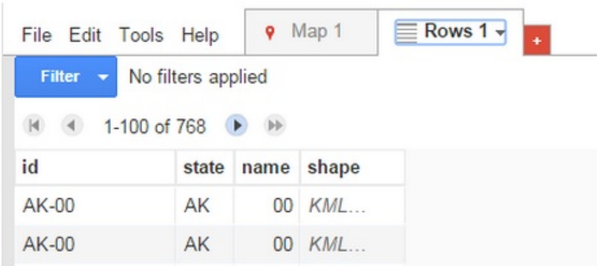
[Congress](https://www.google.com/fusiontables/DataSource?docid=1e23hP8OlwqtiG-ijGn6kLbj-9lgCpeCJkudubQ) (https://www.google.com/fusiontables/DataSource?docid=1e23hP8OlwqtiG-ijGn6kLbj-9lgCpeCJkudubQ) (this is an online open data of Google Fusion Table): (https://www.google.com/fusiontables/DataSource?docid=1e23hP8OlwqtiG-ijGn6kLbj-9lgCpeCJkudubQ)Congressional districts can change every 10 years as a result of the census, and they aren't always built in to mapping tools. Click any district shape to find out more:



...everything is bigger in Texas (except Alaska)

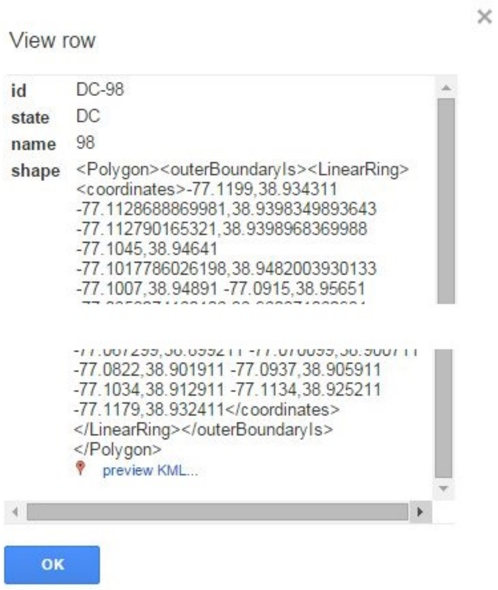
Step 2

Switch to 'Rows 1' view to see the data



(https://www.alexandriarepository.org/wp-content/uploads/20151006084800/GFT9a.jpg)

Why are there so many records (768)?
Which state has the most (use filter)? The least?
Open one of the KML records to explore (by double click on a row), below is shown first and last, try (Washington) DC (it's small):



(https://www.alexandriarepository.org/wp-content/uploads/20151006083217/GFT10.jpg)

Try the 'preview KML...' (shown above)
Which columns are likely contenders for a fuse (or join) with another table?

Step 3

The other table is Household heating by Congressional District - 2008 (another online open data of Google Fusion Table).
https://www.google.com/fusiontables/data?docid=17c8um48aRkljF4bWnxrLJt1enW2_MfVU33Qza-o#rows:id=1

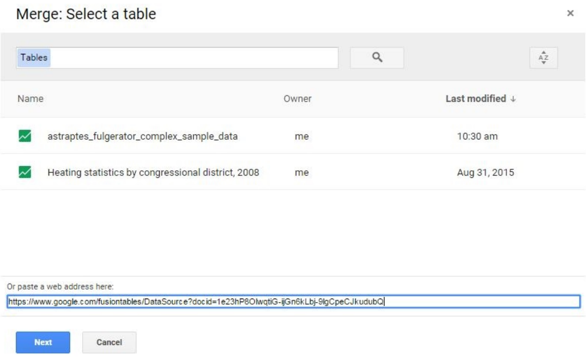
Open & examine, note the 'Two-Digit District' code and pick a column from the other data that is of interest, in this case 'mobile homes'.
Are these typically an indicator of mobility, lower income, lifestyle, other?
Where do you expect them to be clustered (possibly not Alaska - too cold)?

(Both of these tables are defined as exportable and public, so you can find them by searching in Fusion Tables public data).

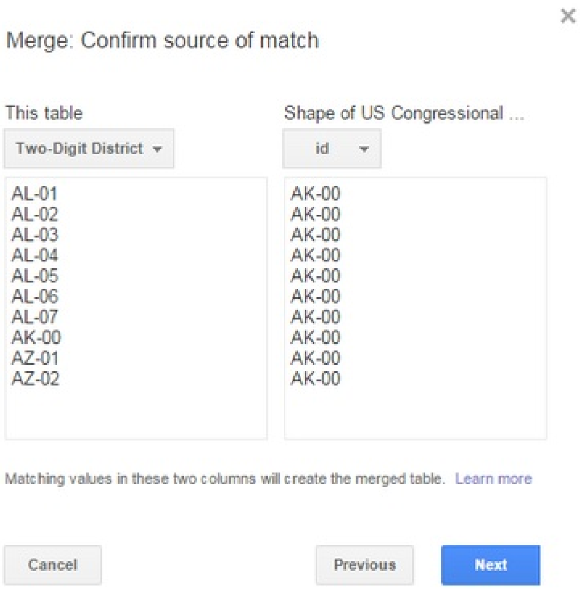
Step 4. Combine the two tables into a new table

1. Open the Home Heating statistics table (if it isn't open already).
2. Click File > Merge. (If inactive, sign in first.)

3. Copy the URL of the Congressional boundaries table (US Congressional District Boundaries, 110th Congress (https://www.google.com/fusiontables/DataSource?docid=1e23hP8OlwqtiG-ijGn6kLbj-9lgCpeCJkudubQ)).
4. Paste it in the "Or paste a web address here" box and click Next.

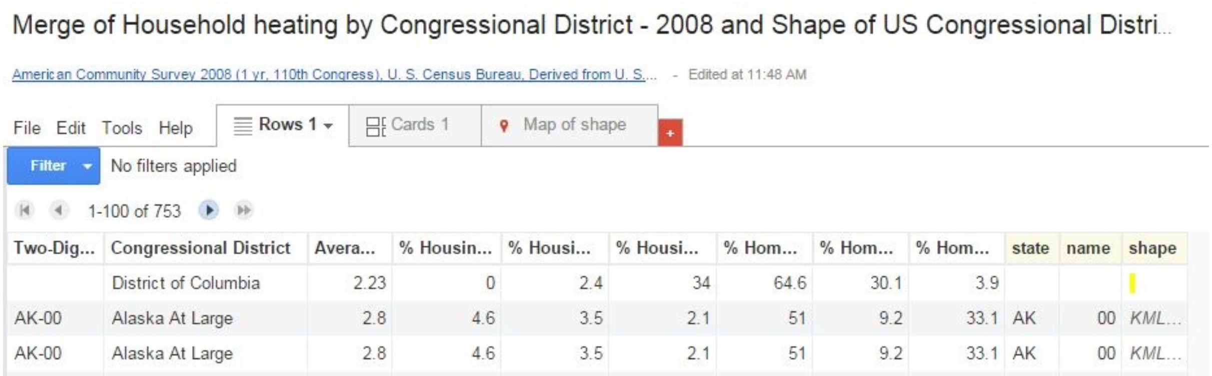


5. Specify the column that both tables have in common:
   On the left, select "**Two-digit District**"
   On the right, "**id**" is already selected.



(https://www.alexandriarepository.org/wp-content/uploads/20151006091202/GFT13a.jpg)

Click 'Next', then 'Merge'
Take a moment to look at your new table ('View table'):



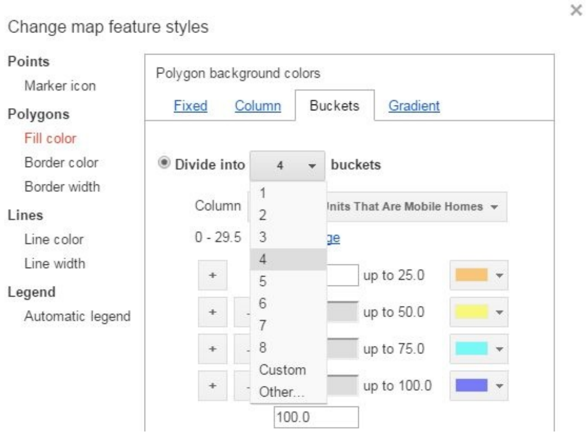
Two-Dig...	Congressional District	Avera...	% Housin...	% Housi...	% Housi...	% Hom...	% Hom...	% Hom...	state	name	shape
	District of Columbia	2.23	0	2.4	34	64.6	30.1	3.9			
AK-00	Alaska At Large	2.8	4.6	3.5	2.1	51	9.2	33.1	AK	00	KML...
AK-00	Alaska At Large	2.8	4.6	3.5	2.1	51	9.2	33.1	AK	00	KML...

The column headers from different source tables have different background colors (pale yellow, far right). Both sources are attributed in the table header. The lineage (history) of the data in the merged table is available in greater detail (see: 'File' > 'About this table'). The new combined table is *not* a copy: if the underlying data is changed, the merged table shows the changes too. Thus if the data owner finds an error and corrects the original data in Fusion Tables, the correction propagates.

Step 5. Customize the map display

Now use some data to determine the display color for each district.

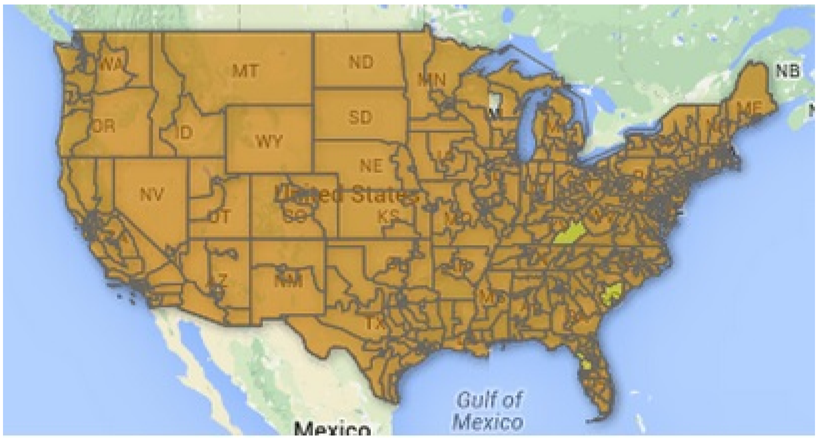
1. Click the 'Map of shape' tab (if it isn't visible and you are not using Chrome as your browser, try using Chrome).
2. Click the 'Change feature styles' button.
3. Within the Map styles dialog, under 'Polygons', click 'Fill color'.
4. Click 'Buckets' on the Polygon background colors panel.



(https://www.alexandriarepository.org/wp-content/uploads/20151006083229/GFT13.jpg)

5. Select the number of buckets from the pull-down menu. For our example, choose 4.
6. Pick the column with the data to map, e.g. '..Mobile Homes'
7. *Optional*: Click any color and select a new color from the color picker.
   Click 'Save'.
8. Look at your map! (note - you may need to zoom in a little to see the states' shapes).

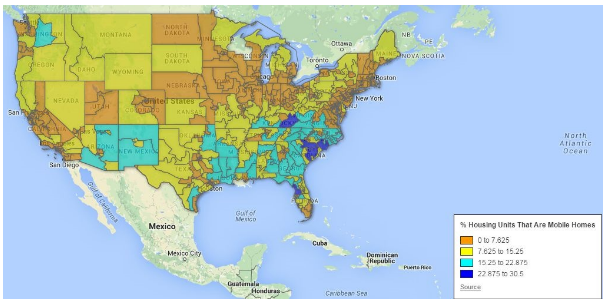
These colours aren't showing much:



Return to the 'Map of shape' tab -> Change this map -> Change Feature styles -> then change the colours via 'Use this range', then 'Save'

What's the difference?

Now go back and add a legend:



(https://www.alexandriarepository.org/wp-content/uploads/20151006083232/GFT14.jpg)

So mobile homes are concentrated in the SouthEast.
What's going on in Washington state?
Return to the data and find out the actual numbers, compare Washington to it's neighbours.

More customization ideas:

- Click a map polygon to see the default info window. You can [customize map info windows](http://www.google.com/support/fusiontables/bin/answer.py?hl=en&answer=171216) (http://www.google.com/support/fusiontables/bin/answer.py?hl=en&answer=171216).
- Use summaries to find out more about your data:
  1. Open a Summary tab.
  2. Choose a column to summarize by.
  3. Choose a column to get more details about and check the "average," "minimum" and "maximum" check boxes.
  4. Click Save.
- Set [filters](https://support.google.com/fusiontables/bin/answer.py?hl=en&answer=171203&topic=2572108&ctx=topic) (https://support.google.com/fusiontables/bin/answer.py?hl=en&answer=171203&topic=2572108&ctx=topic) to narrow your data

(based on https://support.google.com/fusiontables/answer/1032332)

Summary

So you've seen how to overlay Google Maps with data (butterflies) using Google-Fusion Tables (GFT), and, because the 'F' in GFT is for Fusion, the joining, or fusing of tables for geographic data visualisation, i.e. joining databases together (and laying them over a map).

How do you rate GFT as a visualisation tool, as a fusion tool?

Can GFT do standard charts (e.g. scatter, bar, pie)?

Still remember how to "merge" tables in Tableau? and creating choropleth map in R in [Introduction to data exploration and visualisation](https://www.alexandriarepository.org/syllabus/data-exploration/) (https://www.alexandriarepository.org/syllabus/data-exploration/).

Which do you prefer and why?

5
Activity: Maps & data with R

Spatial Visualization with R AKA maps

For a long time, R has had a relatively simple mechanism, via the `maps` package, for making simple outlines of maps and plotting latitude & longitude **points** and **paths** on them.

Now there are a range of options: e.g. `rgdal`, `rgeos`, `rmaps`, `maptools`, `mapdata` (also interactive map options e.g. `shiny`, D3, `leaflet`).
We're going to look at the following packages: `maps`, `ggmap` & `mapproj` (and a bit of `geosphere`):

- **`maps`**; https://cran.r-project.org/web/packages/maps/maps.pdf
- **ggmap**; enables visualization by combining the spatial information of static maps from Google Maps, OpenStreetMap, Stamen Maps or CloudMade Maps with the layered grammar of graphics implementation of ggplot2.
  Also access the Google Geocoding, Distance Matrix, and Directions APIs.(brought to you by the creator of ggplot, Hadley Wickham, see e.g. https://journal.r-project.org/archive/2013-1/kahle-wickham.pdf)"Since ggplot2 is an implementation of the layered grammar of graphics... ***the coordinate system is fixed to the Mercator projection***"http://stat405.had.co.nz/ggmap.pdf
- **`mapproj`**; adds more projections (dozens) see: https://cran.r-project.org/web/packages/mapproj/mapproj.pdf

Start with the map library and a map

```
library( maps )
map ("nz")# what a cute country
```

Location, location, location

Here, we are going to introduce 3 different ways to **define a location** in ggmap.

```
library(ggmap)# load ggmap

# Define location 3 ways
# 1. location/address
myLocation1 <- "Melbourne"
myLocation1

# 2. lat/long
myLocation2 <- c(lon=-95.3632715, lat=29.7632836)# not "Melbourne"
myLocation2

# 3. an area or bounding box (4 points), lower left lon, lower left lat, upper
```

```
right lon, upper right lat
# (a little glitchy for google maps)
myLocation3 <- c(-130, 30, -105, 50)
myLocation3
```

Latitude & longitude

One important geo-feature for geo-related applications is getting latitude and longitude of a given address (country, city, suburb, street and etc.).
As once you have the location, you will know where to place your visualisation for this data.

```
# Convert location/address to its lat/long coordinates:
geocode("Melbourne") # or...
geocode(myLocation1) # confirm this using e.g.
http://www.latlong.net/Show-Latitude-Longitude.html
# Yes, Melbourne is where it's supposed to be in, in Australia
# longitude 144.9633
# latitude -37.814
```

So now let's see a map based on location. There are 4 map sources (online services), with multiple map types:

1. stamen: e.g. "terrain", "toner", "watercolor" etc.
2. google: "roadmap", "terrain", "satellite", "hybrid"
3. osm: open street map
4. cloudmade: 1000s of maps, but you need an api key

In R, try:

```
?get_stamenmap # or help(get_stamenmap) also try ?get_googlemap,
?get_openstreetmap and ?get_cloudmademap
```

to see:

Arguments

bbox	a bounding box in the format c(lowerleftlon, lowerleftlat, upperrightlon, upperrightlat).
zoom	a zoom level
maptype	terrain, terrain-background, terrain-labels, terrain-lines, toner, toner-2010, toner-2011, toner-background, toner-hybrid, toner-labels, toner-lines, toner-lite, or watercolor.
crop	crop raw map tiles to specified bounding box
messaging	turn messaging on/off
urlonly	return url only
color	color or black-and-white
force	if the map is on file, should a new map be looked up?
where	where should the file drawer be located (without terminating "/")
...	...

The get_map function provides a general approach for quickly obtaining maps from multiple sources.

```
myMap <- get_map(location = myLocation1, source = "stamen", maptype =
```

```
"watercolor")
ggmap(myMap)
```

Google Maps

Let's try another different source.

```
myMap <- get_map(location = myLocation1, source="google", color="bw")
ggmap(myMap)
```

Let's also play with the `maptype`.
Please note, you may also need to change the "`maptype`" parameters.
Different source support different map types. Use `?get_googlemap` to check.

```
myMap <- get_map(location = myLocation1, source="google", maptype="terrain",
zoom = 10)
ggmap(myMap)
```

Try some projections

mercator: equally spaced straight meridians, conformal, straight compass courses

then compare with e.g. albers (named after Heinrich C. Albers), is a conic, equal area map projection that uses two standard parallels
https://en.wikipedia.org/wiki/Albers_projection

```
require(mapproj)
m <- map("usa", plot = FALSE) # get map data (lat &amp;amp; lon for boundaries
in this case)

map(m, project = "mercator") # try mercator first
map.grid(m) # draw graticules

map(m, project = "albers", par=c(39,45)) # change the projection to albers
map.grid(m) # draw graticules to compare more easily
```

Let's move on to the whole world.

```
# get unprojected world limits
m <- map('world', plot = FALSE)
# center on New York
map(m, proj = 'azequalarea', orient = c(41,-74,0))
map.grid(m, col = 2) # draw graticules
```

Your turn, rotate the map to show Australia (using R).

'X' marker on the map

```
map(m, proj = 'orth', orient = c(41,-74,0))
map.grid(m, col = 2, nx = 6, ny = 5, label = FALSE, lty = 2)
points(
  mapproject(
    list(y = 41, x = -74)
  ),
  col = 3,
  pch = "x",
  cex = 2
)# centre on NY
```

Your turn, centre on Melbourne.

Label the map

```
map("state", proj='bonne', param=45)
data(state)
text(
  mapproject(
    state.center
  ),
  state.abb
)
```

You may also want to try:

```
map("state",proj='bonne', param=45)
text(
  mapproject(
    state.center,
    proj='bonne',
    param=45
  ),
  state.abb
)
```

However, this does not work.

This is because, the default **orientation** for `map` and `mapproject` are different.

Data on a map using quick map plot

Let's look at the data first. We are going to use the `ggmap` built-in data set **`crime`**.

```
help(crime)
head(crime)
```

This data set is pretty large, here, we will choose a subset from it and plot.

```
murder <- subset(crime, offense == "murder")
qmplot(lon, lat,
  data = murder,
  colour = I('red'),
  size = I(3),
  darken = .3
)
```

Choropleth Map

We are going to create a choropleth map of unemployed rate of US.

Two data sets will be used here unemp and `county.fips`.

Data always should go first, have a look at data sets first.

```
help(unemp)
head(unemp)

help(county.fips)
head(county.fips)
```

Let's pre processing the data.

We want to split the unemployed rate into 7 intervals ("<2%","2-4%","4-6%","6-8%","8-10%",">10%").

```
# use the version installed with maps library!
data(unemp)

# set up intervals
Intervals <-as.numeric(
  cut(
    unemp$unemp,
    c(0,2,4,6,8,10,100)
  )
)
```

Then we need to match unemployment data to map regions by fips codes.

```
data(county.fips)
Matches <- Intervals[
  match(
    county.fips$fips,
```

```
        unemp$fips
  )
]
```

After that, we can prepare the color schema and plot the map.

```
colors <- c("#ffffd4","#fee391","#fec44f","#fe9929","#d95f0e","#993404")
# draw map
map("county",
  col = colors[Matches],
  fill = TRUE,
  resolution = 0,
  lty = 0,
  projection = "polyconic"
)
```

State boundaries will make it better.

```
# draw state boundaries
map("state",
  col = "purple",
  fill = FALSE,
  add = TRUE,
  lty = 1,
  lwd = 0.3,
  projection = "polyconic"
)
```

Never forget the title and legend.

```
# add title and legend
title("Unemployment by county, 2009")

Legend <- c("<2%","2-4%","4-6%","6-8%","8-10%",">10%")
legend("topright", Legend, horiz = TRUE, fill = colors)
```

Your turn!

Change the intervals to ("<5%","5-10%","10-15%","15-20%","20-25%",">25%").

Flow Map

Based on http://flowingdata.com/2011/05/11/how-to-map-connections-with-great-circles/
Let's look at how to draw a line on the map.
Note: the shortest path between two locations is usually not **a straight line** on a map, because of the map projection. The shortest path is always the "great circle" that passes through the two points. This is not the same as the path travelled by a vehicle travelling on a fixed bearing which is what is shown on as a straight line using the Mercator projection.

```
library(geosphere)
```

```
map("state")
lat_ca <- 39.164141
lon_ca <- -121.640625

lat_me <- 45.213004
lon_me <- -68.906250

inter <- gcIntermediate(
  c(lon_ca,lat_ca),
  c(lon_me,lat_me),
  n = 50,
  addStartEnd=TRUE
)
lines(inter)
```

Now we can draw **lines** of flights.

Again, let's look at the data first.

```
airports <-
read.csv("http://datasets.flowingdata.com/tuts/maparcs/airports.csv", header =
TRUE)
flights <- read.csv("http://datasets.flowingdata.com/tuts/maparcs/flights.csv",
header = TRUE, as.is = TRUE)

head(airports)
head(flights)
```

Plot the map and flights.

```
# create a world map and limited it to around US areas.
xlim <- c(-171.738281, -56.601563)
ylim <- c(12.039321, 71.856229)
map(
  "world",
  col="#f2f2f2",
  fill=TRUE,
  bg="white",
  lwd=0.05,
  xlim=xlim,
  ylim=ylim
)

fsub <- flights[flights$airline == "AA",]

for(j in 1:length(fsub$airline))
{
  air1 <- airports[
    airports$iata == fsub[
      j,]$airport1,]
  air2 <- airports[
    airports$iata == fsub[
```

```
      j,]$airport2,]
  inter <- gcIntermediate(
    c(
      air1[1,]$long,
      air1[1,]$lat
    ),
    c(
      air2[1,]$long,
      air2[1,]$lat
    ),
    n = 100,
    addStartEnd = TRUE
  )
  lines(inter, col="black", lwd=0.8)
}
```

So what can/can't you do with ggmap?
Compare R tools with other options, e.g. GIS tools, also Tableau Public, Google FusionTables.

6
Activity: Shape files & maps with R

Shapefiles

A shapefile, sometimes called an ESRI shapefile (aka Environmental Systems Research Institute) is a format for storing the location, shape, and attributes of geographic features. It is stored as a set of related files (about a dozen).

We require a minimum of 3 files:

- `.shp - shape format; the feature geometry itself`
- `.shx - shape index format; a positional index of the feature geometry to allow seeking forwards and backwards quickly`
- `.dbf - attribute format; columnar attributes for each shape, in dBase IV format (how old is that!)`

https://en.wikipedia.org/wiki/Shapefile

The files (.shp, .shx, .dbf) should all have the same prefix (e.g. NZ.shp, NZ.shx, NZ.dbf). We require 3 sets of files in this activity, you can download them from here:
NZ (https://www.alexandriarepository.org/wp-content/uploads/20170221140142/NZL_adm1.zip)
World (https://www.alexandriarepository.org/wp-content/uploads/20170221140313/ne_110m_land1.zip)
US parks (https://www.alexandriarepository.org/wp-content/uploads/20160302132619/ne_10m_parks_and_protected_lands.zip)
`maptools` library is one option to load shapefiles in R.
Install the library first.
Load the library.

```
library(maptools)
```

Look carefully at your RStudio console.

There might be an warning:

```
> library(maptools)
Checking rgeos availability: FALSE
        Note: when rgeos is not available, polygon geometry     computations in maptools depend on gpclib,
        which has a restricted licence. It is disabled by default;
        to enable gpclib, type gpclibPermit()
```

So, according to the prompt message, you should run

```
gpclibPermit()
```

If it is still not work, that means you need to install another package.

Remember the other way to install a package?

```
install.packages("gpclib")
gpclibPermit()
```

```
> install.packages("gpclib")
trying URL 'https://cran.rstudio.com/bin/macosx/mavericks/contrib/3.3/gpclib_1.5-5.tgz'
Content type 'application/x-gzip' length 121430 bytes (118 KB)
==================================================
downloaded 118 KB


The downloaded binary packages are in
	/var/folders/3t/613lt_895sj3zj8lxd7sg24c0000gr/T//Rtmpg3pL0B/downloaded_packages
> gpclibPermit()
[1] TRUE
Warning message:
In gpclibPermit() :
  support for gpclib will be withdrawn from maptools at the next major release
```

Load the library again, and you should be ready to use `maptools` library.

Unzip your downloaded NZ shapefile and place the 3 files (NZL_adm0.shp, NZL_adm0.shx and NZL_adm0.dbf) into your **working directory**.
Now everything is ready, do not wait, just plot it:

```
nz <- readShapeSpatial("NZL_adm0")
plot(nz)
```

Be patient, takes a bit time.

Also, the plot is a bit...unexpected.

Any way, move on to the **world**!

Move the 3 files (ne_110m_land.shp, ne_110m_land.shx and ne_110m_land.dbf) to your **working directory**.

```
world <- readShapeSpatial("ne_110m_land")
plot(world)
```

Let's plot it in another way (with ggplot).

```
library(ggplot2)
world_shp <- readShapePoly("ne_110m_land.shp")
ggplot(
  world_shp,
  aes(
    x=long,
    y=lat,
    group=group
  )
) +
geom_path()
```

Head into the detail of `world_shp`.

You may get confused about the stored structure.
You can use `fortify()` to transfer it to our familiar tabular format.
Note: this is **not necessary**.

```
head(world_shp)
world_map <- fortify(world_shp) # convert into a tabular structure
head(world_map)
```

And use the tabular format to plot.

```
ggplot(
  world_map,
  aes(
    x=long,
    y=lat,
    group=group
  )
) +
geom_path()
```

Let's do a shape file on a map

National parks of America on top of the West coast.

From: http://blog.mollietaylor.com/2013/02/shapefiles-in-r.html

First, put the 3 files (ne_10m_parks_and_protected_lands_area.shp, ne_10m_parks_and_protected_lands_area.shx and ne_10m_parks_and_protected_lands_area.dbf) into your **working directory**.

and read them.

```
library(ggmap) # Load the shapes and transform
library(maptools)

area <- readShapePoly("ne_10m_parks_and_protected_lands_area.shp")
area.points <- fortify(area) # transform
```

Now let's have a look at how the parks distribute.

```
# Add some colour
library(RColorBrewer)
colors <- brewer.pal(9,"BuGn")

# Get the underlying map, it may take a while to get (from google), patience...
mapImage <- get_map(
  location = c(lon=-118,lat =37.5),
  color = "color",
  source="google",
  maptype = "terrain",
  zoom =6
)

# Put the shapes on top of the map
ggmap(mapImage) +
  geom_polygon(
    aes(x=long, y=lat, group=group),
    data = area.points,
    color = colors[9],
    fill = colors[6],
    alpha =0.5
  ) +
  labs(x="Longitude", y="Latitude")
```

Step by step.

```
# Plot the base map
plot(mapImage)
```

Then the parks without the base map.

```
# And the parks without the map...
p <- ggplot()
# a blank
p +
  geom_polygon(
    aes(x=long, y=lat, group=group),
    data=area.points,
    color=colors[9],
    fill=colors[6],
    alpha =0.5
  )+
  labs(x="Longitude", y="Latitude")
```

And we can stack them layer by layer.

We still forget something.

Remember the unexpected NZ at the beginning?
Why does a huge shape file result in such a small plot (NZ)?

(one theory is that there are lots of little islands, another is that there are too many sheep)

Don't forget to restore NZ to it's former glory

```
nz <- map_data("nz")
#Prepare a map of NZ
nzmap <- ggplot(nz, aes(x = long, y = lat, group = group)) +
geom_polygon(fill = "gold", colour = "gold")
# Plot it in cartesian coordinates
# nzmap
# With correct mercator projection
nzmap + coord_map()
# With the aspect ratio approximation
# nzmap + coord_quickmap()
```

What formats are these built-in maps, e.g. map_data("nz"), is it a .shp?

How old is this ESRI format?
What are some more recent alternatives?
Can you open just the .shp files (and view shapes) - in R? Some other way?

7
Activity: Leaflet with R

A. Introduction

Leaflet is an open-source JavaScript library for **interactive** maps.

A basic interactive map (pan & zoom & markers & etc...) can be created very easily with `leaflet`.

The `leaflet` R package allows you to use Leaflet interactive maps in R in an easy way.

We start by installing the package "`leaflet`".

Let's start mapping....

The very first map with leaflet.

```
m <- leaflet() %>%
  setView(lng = 145.0431, lat = -37.8773, zoom = 15) %>%
  addTiles()
m
```

There is a very "modern" way to use leaflet with the `%>%` operation.

The basic idea is to use **the result of the *previous function*** as **the first parameter of the *next function***.

Check [magrittr pipe operator](https://github.com/tidyverse/magrittr) (https://github.com/tidyverse/magrittr) for details.

Or we can do it in the old fashioned way:

```
m <- leaflet()
m <- setView(m, lng = 145.0431, lat = -37.8773, zoom = 15)
m <- addTiles(m)
m
```

You may already notice, the map is plotted in the "Viewer" tab instead of the normal "Plots" tab.

This is because the leaflet R package is still using the leaflet javascript package and the "Viewer" tab is basically a small web browser.

You can also customise your tile.

```
m %>% addProviderTiles("Stamen.Toner")
```

use `help(addProviderTiles)` to check other available tiles.

B. Marker

The data is ready for you in: vet_schools_in_victoria
(https://www.alexandriarepository.org/wp-content/uploads/20170220154101/vet_schools_in_victoria.txt).

Load it as usual and have a look.

```
library(leaflet)
data <- read.csv("vet_schools_in_victoria.txt")
head(data)
```

This data set is very large.

Let's place the first 30 records on the map first.

```
leaflet(data = data[1:30, ]) %>% addTiles() %>%
  addMarkers(~longitude, ~latitude, popup = ~as.character(location))
```

Click on the markers, find out what's the text about.

Try the whole data set.

```
leaflet(data = data) %>% addTiles() %>%
  addMarkers(~longitude, ~latitude, popup = ~as.character(location))
```

That's terrible...

But our map is interactive, we should be able to cluster the data and allow the user to zoom into the details.

```
leaflet(data = data) %>% addTiles() %>%
  addMarkers(
    ~longitude,
    ~latitude,
    popup = ~as.character(location),
    clusterOptions = markerClusterOptions()
  )
```

C. Choropleth Map

Download our familiar data set again: Household-heating-by-State-2008.csv
(https://www.alexandriarepository.org/wp-content/uploads/20151006100756/Household-heating-by-State-2008.csv).

Let's pre-process the data as usual.

```
data <- read.csv("Household-heating-by-State-2008.csv", header=T)
names(data)[4] <- "MobileHomes"
ag <- aggregate(MobileHomes ~ States, FUN = mean, data = data)
ag$States <- tolower(ag$States)
```

Let's prepare the map data and link the two data sets.

```
library(maps)
mapStates <- map("state", fill = TRUE, plot = FALSE)
rates <- ag$MobileHomes[match(mapStates$names, ag$States)] # find the related
rate for each state
```

Now, it is time to use `leaflet`.

```
library(leaflet)
cpal <- colorNumeric("Blues", rates) # prepare the color mapping
leaflet(mapStates) %>% # create a blank canvas
  addTiles() %>% # add tile
  addPolygons( # draw polygons on top of the base map (tile)
    stroke = FALSE,
    smoothFactor = 0.2,
    fillOpacity = 1,
    color = ~cpal(rates) # use the rate of each state to find the correct color
  )
```

You may also notice some parts are not colored.

Let's check out why.

```
mapStates$names
ag$States
```

```
> mapStates$names
 [1] "alabama"                    "arizona"                          "arkansas"
 [4] "california"                 "colorado"                         "connecticut"
 [7] "delaware"                   "district of columbia"             "florida"
[10] "georgia"                    "idaho"                            "illinois"
[13] "indiana"                    "iowa"                             "kansas"
[16] "kentucky"                   "louisiana"                        "maine"
[19] "maryland"                   "massachusetts:martha's vineyard"  "massachusetts:main"
[22] "massachusetts:nantucket"    "michigan:north"                   "michigan:south"
[25] "minnesota"                  "mississippi"                      "missouri"
[28] "montana"                    "nebraska"                         "nevada"
[31] "new hampshire"              "new jersey"                       "new mexico"
[34] "new york:manhattan"         "new york:main"                    "new york:staten island"
[37] "new york:long island"       "north carolina:knotts"            "north carolina:main"
[40] "north carolina:spit"        "north dakota"                     "ohio"
[43] "oklahoma"                   "oregon"                           "pennsylvania"
[46] "rhode island"               "south carolina"                   "south dakota"
[49] "tennessee"                  "texas"                            "utah"
[52] "vermont"                    "virginia:chesapeake"              "virginia:chincoteague"
[55] "virginia:main"              "washington:san juan island"       "washington:lopez island"
[58] "washington:orcas island"    "washington:whidbey island"        "washington:main"
[61] "west virginia"              "wisconsin"                        "wyoming"
> ag$States
 [1] "#n/a"           "alabama"        "alaska"         "arizona"        "arkansas"       "california"
 [7] "colorado"       "connecticut"    "delaware"       "florida"        "georgia"        "hawaii"
[13] "idaho"          "illinois"       "indiana"        "iowa"           "kansas"         "kentucky"
[19] "louisiana"      "maine"          "maryland"       "massachusetts"  "michigan"       "minnesota"
[25] "mississippi"    "missouri"       "montana"        "nebraska"       "nevada"         "new hampshire"
[31] "new jersey"     "new mexico"     "new york"       "north carolina" "north dakota"   "ohio"
[37] "oklahoma"       "oregon"         "pennsylvania"   "rhode island"   "south carolina" "south dakota"
[43] "tennessee"      "texas"          "utah"           "vermont"        "virginia"       "washington"
[49] "west virginia"  "wisconsin"      "wyoming"
```

As you can see, "washington" is split into different parts in `mapState`, and so the strings do not match.

Data processing again.

```
# split the string with : as seperator
spliteNames <- strsplit(mapStates$names, ":")
# get first part of the origin string;
# e.g. get washington from washington:san juan island
firstPartNames <- lapply(spliteNames, function(x) x[1])
rates <- ag$MobileHomes[match(firstPartNames, ag$States)]
```

Now plot again.

```
cpal <- colorNumeric("Blues", rates) # prepare the color mapping
leaflet(mapStates) %>% # create a blank canvas
  addTiles() %>% # add tile
  addPolygons( # draw polygons on top of the base map (tile)
    stroke = FALSE,
    smoothFactor = 0.2,
    fillOpacity = 1,
    color = ~cpal(rates) # use the rate of each state to find the correct color
  )
```

Compare it with creating choropleth map in **R** using in previous activities, **Google Fusion Table** and **Tableau**. Which one do you prefer and **WHY**?
Here, we are using the built-in map data `map("state", fill = TRUE, plot = FALSE)`, if you

cannot find your map data in the built-in maps, `leaflet` can also handle shapefiles.

You will need `rgdal` library to read your shapefile and it is available at: World shape file (https://www.alexandriarepository.org/wp-content/uploads/20170221105200/ne_50m_admin_0_countries.zip).

Unzip it and put the very important 3 files (ne_50m_admin_0_countries.shp & shx & dbf) into your working directory.

Also we need the data for colors. Here we provide the GPD data for countries (https://www.alexandriarepository.org/wp-content/uploads/20170221111801/WorldGDP.txt).

Reading the data first.

```
library(rgdal)
world_map <- readOGR("ne_50m_admin_0_countries.shp")
gdp_data <- read.csv("WorldGDP.txt")
```

Match the gdp to each country.

```
rates <- gdp_data$GDP[match(world_map$admin, gdp_data$Name)]
```

Create the map

```
library(leaflet)
qpal <- colorQuantile("Blues", rates, 12) # prepare the color mapping

leaflet(world_map) %>% # create a blank canvas
  addTiles() %>% # add tile
  addPolygons( # draw polygons on top of the base map (tile)
    stroke = FALSE,
    smoothFactor = 0.2,
    fillOpacity = 1,
    color = ~qpal(rates) # use the rate of each state to find the correct color
  )
```

If you read carefully enough, you may find two different functions are used for color mapping.

- colorQuantile
- colorNumeric

Investigate what's the difference between them.

There is also some mismatching...

The reason might be the same as our previous US one, can you fix it?

Also be careful with your own data!

D. Work with Shiny

`Leaflet` can also integrate into your Shiny app easily.

Here, we are also introducing a new way to write a Shiny app, with all the code in one file.

```
library(shiny)
library(leaflet)

ui <- fluidPage(
  leafletOutput("mymap"), # create map canvas on the page
  actionButton("recalc", "New points") # create a button, and bind it to the
recalc event
)

server <- function(input, output, session) {
  points <- eventReactive(input$recalc, { # event handle, in this case for click
event
    cbind(rnorm(40) * 3 + 145.0431, rnorm(40) -37.8773) # calculate normal
distribution random points around Melbourne
  }, ignoreNULL = FALSE)
  output$mymap <- renderLeaflet({ # create leaflet map
    leaflet() %>%
      addTiles() %>%
      addMarkers(data = points()) # use the random generated points as markers
on the map
  })
}

shinyApp(ui, server)
```

Create a new file and paste the above code to your RStudio (and understand it!).

After you save your file, you will find the magic button (Run App) at the right button of your code.

More details on the integration with Shiny is available at: https://bhaskarvk.github.io/leaflet/shiny.html (the D section is also based on this material).