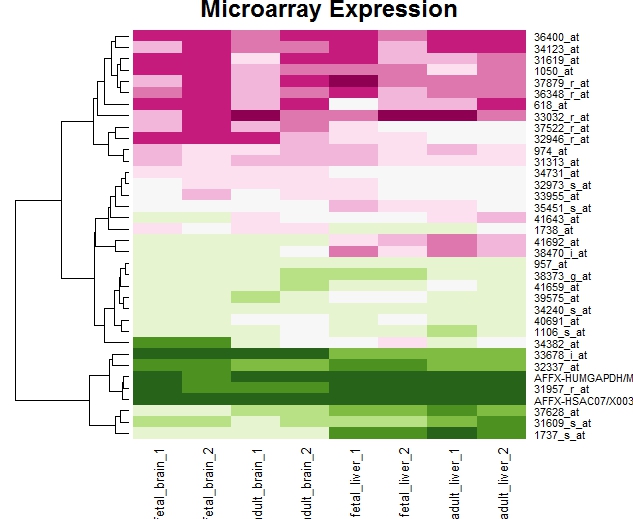
Heatmaps are tables that use colors instead of numbers to represent data. Especially for large datasets, heatmaps can be a useful way to visualize data, making it easy to pick out high and low values, identify patterns, and locate outliers. Heatmaps are often used to visualize gene expression data, but can be used for almost any type of tabular or array data. In this class, we’ll learn how to use RStudio, a free software package, to create customized heatmaps. Some experience with using R is helpful, but not necessary. Attendees will learn how to:

* Decide when a heatmap is the most appropriate visualization for data and interpret heatmaps;
* Process data in RStudio so it is in the proper format for creating a heatmap;
* Create a heatmap and customize it by selecting colors, creating labels, and saving the file in different formats.

Participants in the class will learn to make heatmaps similar to this oen:



Mosaic Plots

For extended mosaic plots, use **mosaic(x, condvar=, data=)** where **x** is a table or formula, **condvar=** is an optional conditioning variable, and **data=** specifies a data frame or a table. Include **shade=TRUE** to color the figure, and **legend=TRUE** to display a legend for the Pearson residuals.

# Correlograms

**Correlograms** help us visualize the data in correlation matrices.

In R, correlograms are implimented through the **corrgram(x, order = , panel=, lower.panel=, upper.panel=, text.panel=, diag.panel=)** function in the corrgrampackage.

Options

**x**is a data frame with one observation per row.

**order=TRUE** will cause the variables to be ordered using principal component analysis of the correlation matrix.

**panel**= refers to the off-diagonal panels. You can use**lower.panel=** and **upper.panel**= to choose different options below and above the main diagonal respectively. **text.panel=**and **diag.panel=** refer to the main diagnonal. Allowable parameters are given below.

***off diagonal panels*  
panel.pie** (the filled portion of the pie indicates the magnitude of the correlation)  
**panel.shade** (the depth of the shading indicates the magnitude of the correlation)  
**panel.ellipse** (confidence ellipse and smoothed line)  
**panel.pts** (scatterplot)

***main diagonal panels*  
panel.minmax** (min and max values of the variable)  
**panel.txt** (variable name).

# First Correlogram Example  
library(corrgram)  
corrgram(mtcars, order=TRUE, lower.panel=panel.shade,  
  upper.panel=panel.pie, text.panel=panel.txt,  
  main="Car Milage Data in PC2/PC1 Order")

## Getting started with ggmap

|  |  |  |
| --- | --- | --- |
| 1 | install.packages("ggmap") | |
| 2 | library(ggmap) |

That’s it.

The fastest way to get going is with the qmap class, which stands for “quick map plot”. Play around with the different types of parameter calls to render various plot types.

Some examples to start:

|  |  |
| --- | --- |
| 1 | qmap(location = "boston university") |
| 2 | qmap(location = "boston university", zoom = 14) | |

|  |  |
| --- | --- |
| 3 | qmap(location = "boston university", zoom = 14, source = "osm") |



Here’s how it works: qmap is a wrapper for get\_map and ggmap. get\_map is a smart wrapper that queries the map server of your choosing—Google Maps, OpenStreetMap, or Stamen Maps—and returns a map at a specified location. (Sorry, Apple and Bing Maps fans, there’s no support yet for these APIs.)

As the above example shows, there’s no need to add a specific latitude or longitude. ggmap accepts text search inputs with the “location” parameter when creating a map.

## Visualizing clusters

Spatial plotting capabilities are helpful for those running analytics in certain actuarial science industries like health or car insurance.

For example, let’s visually investigate the number of motor vehicle collisions state by state. To start, I found an Excel file of fatal crashes in 2012 from the Fatality Analysis Reporting System (FARS) Encyclopedia.

(A clean .csv version of the dataset is available in this public project on Domino’s platform for data science.)

After loading the ggmap library, we need to load and clean up the data. The following lines load a CSV file, convert the State column to character data type, and turns the Motor Vehicle collision amounts from integer to double. Lastly, we remove Hawaii and Alaska to get a tighter map view. (Sorry!)

|  |  |
| --- | --- |
| 1 | mydata = read.csv("vehicle-accidents.csv") |
| 2 | mydata$State <- as.character(mydata$State) | |

|  |  |  |
| --- | --- | --- |
| 3 | mydata$MV.Number = as.numeric(mydata$MV.Number) | |
| 4 | mydata = mydata[mydata$State != "Alaska", ] |

|  |  |
| --- | --- |
| 5 | mydata = mydata[mydata$State != "Hawaii", ] |

Next, we use geocode to find a latitude and longitude with theGoogle Maps API using only the character string in mydata$State.

The following is a simple for loop that runs over each state and returns lat/lon coordinates:

|  |  |
| --- | --- |
| 1 | for (i in 1:nrow(mydata)) { |
| 2 | latlon = geocode(mydata[i,1]) | |

|  |  |  |
| --- | --- | --- |
| 3 | mydata$lon[i] = as.numeric(latlon[1]) | |
| 4 | mydata$lat[i] = as.numeric(latlon[2]) |

|  |  |
| --- | --- |
| 5 | } |

Since we aren’t looking at other aspects of the data such as nonmotorist or fixes object collisions, we can create a new data frame to simplify the dataset:

|  |  |  |
| --- | --- | --- |
| 1 | mv\_num\_collisions = data.frame(mydata$MV.Number, mydata$lon, mydata$lat) | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | colnames(mv\_num\_collisions) = c('collisions','lon','lat') |

Now let’s plot the number of collisions per state with varying sizes of circles to see the biggest motor vehicle collision offenders.

We get the geocode for the United States, then create a Google map that covers an area from coast to coast:

|  |  |  |
| --- | --- | --- |
| 1 | usa\_center = as.numeric(geocode("United States")) | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | USAMap = ggmap(get\_googlemap(center=usa\_center, scale=2, zoom=4), extent="normal") |

We use the + operator to add ggplot2 geometric objects and other styling options on top of the map.

The ability to combine ggmap and ggplot2 functionality is a huge advantage for visualizing data with heat maps, contour maps, or other spatial plot types. Most of this overlay capability stems from ggplot2’s geoms, or geometric objects, that determine the shape of the plot being created.

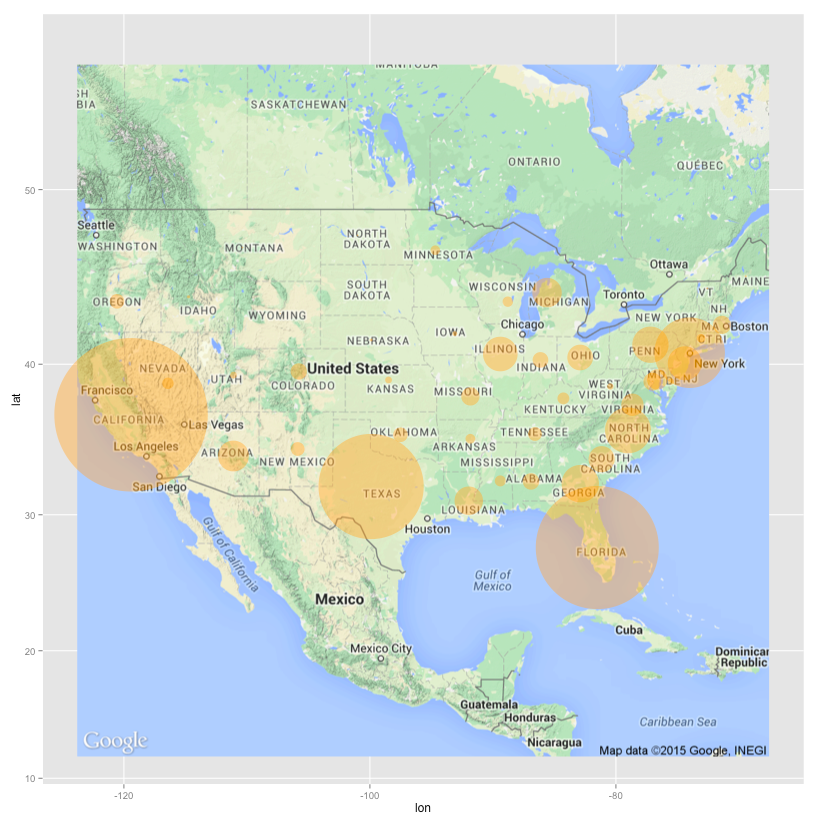
Next we add the geom\_point geom to the map and generate aesthetic mappings with aes that describe how variables in the data are mapped to visual properties (aesthetics) of geoms.

Finally, the size and scale of each circle is based on the minimum and maximum value range of collision amounts per state.

|  |  |
| --- | --- |
| 1 | USAMap + |
| 2 | geom\_point(aes(x=lon, y=lat), data=mv\_num\_collisions, col="orange", alpha=0.4, size=mv\_num\_collisions$collisions\*circle\_scale\_amt) + | |

|  |  |
| --- | --- |
| 3 | scale\_size\_continuous(range=range(mv\_num\_collisions$collisions)) |

Running the **ggmap-demo-circles.R** script in Domino results in a nice map of the biggest offenders: California, Florida, Texas and New York.



Shocking? No. Fun? Yes!

## Heat maps

Let’s try one more plot type—the heat map. Continuing with the theme of data visualization for insurance insights, the next dataset looks at concentration of homes in a region and when those homes were built.

We can use get\_map to download the base map, then draw a gg\_map on top. Then we add:

* geom\_density2d: Perform a 2D kernel density estimation using kde2d and display the results with contours.
* stat\_density2d: 2D density estimation
* scale\_alpha: Sets alpha value for transparency.

Running the ggmap-demo-heat.R script gives the result:

