Broom Spatial R Class

Frank Davenport

June 19, 2012

1 Get Your R On

This preliminary section will cover some basic details about R.

1.1 Data Structures

There are several ways that data are stored in R. Here are the main ones:

- **Data Frames** The most common format. Similar to a spread sheet. A data.frame() is indexed by rows and columns and store numeric and character data. The data.frame is typically what we use when we read in csv files, do regressions, et cetera.
- **Matrices and Arrays** Similar to data.frames but slightly faster computation wise while sacrificing some of the flexibility in terms of what information can be stored. In R a matrix object is a special case of an array that only has 2 dimensions. IE, an array is n-dimensional matrix while a matrix only has rows and columns (2 dimensions)
- **Lists** The most common and flexible type of R object. A list is simply a collection of other objects. For example a regression object is a list of: 1)Coefficient estimates 2) Standard Errors 3) The Variance/Covariance matrix 4) The design matrix (data) 5) Various measures of fit, et cetera.

We will look at examples of these objects in the next section

1.2 Reading Data in and Out

The most common way to read in data is with the `read.csv()` command. However you can read in virtually any type of text file. Type ?read.table in your console for some examples. If you have really large binary data sets sometimes the `scan()` function is more efficient. Finally using the foreign package you can read in SPSS, STATA, Matlab, SAS, and a host of other data formats from other stat and math software.

Let's read in a basic csv file.

```
# -------------READING DATA IN AND OUT---------------
mydat <-
read.csv("/Users/frankdavenport/Education/R_Work/SVN/broom/data/kenpop89to99.csv")
# mydat<-read.csv('H:/broom/data/kenpop89to99.csv')
```

We can explore the data using the names(), summary(), head(), and tail() commands (we will use these frequently through out the exercise)

```
names(mydat)  #column names

##  [1] "ip89DId"    "ip89DName"  "ADMIN3"     "KEADMN3_ID" "Y89Pop"     "Y89Births"
##  [7] "Y89Brate"   "Y99Pop"     "Y99Births"  "Y99Brate"   "PopChg"     "BrateChg"
```

```r
summary(mydat)  #basic summary information
```

```
##     ip89DId            ip89DName          ADMIN3     KEADMN3_ID       Y89Pop
##  Min.   :1010   Kisii       : 3   KISII      : 2   Min.   : 1.0   Min.   :  57960
##  1st Qu.:3772   Kakamega    : 2   BARINGO    : 1   1st Qu.:12.8   1st Qu.: 222905
##  Median :6010   Kericho     : 2   BOMET      : 1   Median :24.5   Median : 451510
##  Mean   :5207   Machakos    : 2   BUNGOMA    : 1   Mean   :25.5   Mean   : 619710
##  3rd Qu.:7052   Meru        : 2   BUSIA      : 1   3rd Qu.:35.2   3rd Qu.: 947500
##  Max.   :8030   South Nyanza: 2   E. MARAKWET: 1   Max.   :63.0   Max.   :1476500
##                 (Other)     :35   (Other)    :41
##    Y89Births        Y89Brate        Y99Pop          Y99Births        Y99Brate
##  Min.   : 1680   Min.   :22.6   Min.   :  72380   Min.   : 1760   Min.   :19.0
##  1st Qu.: 9350   1st Qu.:33.5   1st Qu.: 392545   1st Qu.:10870   1st Qu.:28.0
##  Median :18270   Median :37.4   Median : 629740   Median :21820   Median :31.0
##  Mean   :23719   Mean   :37.0   Mean   : 872928   Mean   :27562   Mean   :31.6
##  3rd Qu.:39855   3rd Qu.:40.9   3rd Qu.:1384665   3rd Qu.:42140   3rd Qu.:36.4
##  Max.   :57460   Max.   :51.0   Max.   :2363120   Max.   :69380   Max.   :42.9
##
##      PopChg          BrateChg
##  Min.   :-14.0   Min.   :-38.00
##  1st Qu.: 23.8   1st Qu.:-20.00
##  Median : 33.5   Median :-14.00
##  Mean   : 47.7   Mean   :-14.56
##  3rd Qu.: 44.2   3rd Qu.: -6.75
##  Max.   :343.0   Max.   :  0.00
##
```

```r
head(mydat)  #first 6 rows
```

```
##   ip89DId ip89DName    ADMIN3 KEADMN3_ID  Y89Pop Y89Births Y89Brate  Y99Pop Y99Births
## 1    1010   Nairobi   NAIROBI         41 1325620     42560    32.11 2085820     58700
## 2    2010    Kiambu    KIAMBU         38  908120     27720    30.52 1383300     36140
## 3    2020 Kirinyaga KIRINYAGA         29  389440     10980    28.19  452180     10840
## 4    2030   Muranga   MURANGA         36  862540     27940    32.39  737520     16500
## 5    2040 Nyandaura NYANDARUA         22  348520     12520    35.92  468300     13320
## 6    2050     Nyeri     NYERI         26  607980     17540    28.85  644380     14340
##   Y99Brate PopChg BrateChg
## 1    28.14     57      -12
## 2    26.13     52      -14
## 3    23.97     16      -15
## 4    22.37    -14      -31
## 5    28.44     34      -21
## 6    22.25      6      -23
```

```r
tail(mydat)  # last 6 rows
```

```
##    ip89DId   ip89DName      ADMIN3 KEADMN3_ID  Y89Pop Y89Births Y89Brate  Y99Pop
## 43    7120 Uasin-Gishu UASIN GISHU         13  443280     17900    40.38  616240
## 44    7130  West-Pokot  WEST POKOT          5  224640      9440    42.02  309020
## 45    8010      Bugoma     BUNGOMA         11  741940     34600    46.63 1008080
## 46    8020       Busia       BUSIA         16  425380     18640    43.82  547680
## 47    8030    Kakamega      VIHIGA         21 1476500     57460    38.92 2011960
```

```
## 48    8030    Kakamega    KAKAMEGA          14 1476500     57460    38.92 2011960
##    Y99Births Y99Brate PopChg BrateChg
## 43     22260    36.12     39      -11
## 44     12940    41.87     38        0
## 45     43240    42.89     36       -8
## 46     23440    42.80     29       -2
## 47     69380    34.48     36      -11
## 48     69380    34.48     36      -11


# --Write data out
# write.csv('/Users/frankdavenport/Education/R_Work/SVN/broom/data/deletethis.csv')

# --Save a List of Objects in Your R Workspace
# save(mydat,someobject,anotherobject,file='Allmystuff.Rdata')
```

We will go over ways to index and subscript data.frames later on in the exercise. For now lets do a basic regression so you can see an example of a list

1.3 Basic Regression (and an example of lists)

We use the `lm()` command to do a basic linear regression. The ~ symbol separates the left and right hand sides of the equation and we use '+' to separate terms and '*' to specify interactions.

```
# -------------REGRESSION AND LISTS---------------

myreg <- lm(Y99Pop ~ Y89Births + Y89Brate, data = mydat)  #Regress the Population in 1999
on the population and birthrate in 1989

myreg

##
## Call:
## lm(formula = Y99Pop ~ Y89Births + Y89Brate, data = mydat)
##
## Coefficients:
## (Intercept)    Y89Births     Y89Brate
##      502593           38       -14369
##
```

A regression object is an example of a list. We can use the `names()` command to see what the list contains. We can use the summary() command to get a standard regression output (coefficients, standard errors, et cetera) and we can also create a new object that contains all the elements of a regression summary.

```
# -------------EXPLORE A REGRESSION OBJECT---------------

names(myreg)  #get the names of the items in the regression object

##  [1] "coefficients"  "residuals"     "effects"       "rank"          "fitted.values"
##  [6] "assign"        "qr"            "df.residual"   "xlevels"       "call"
## [11] "terms"         "model"


summary(myreg)  #print out the key information
```

```
##
## Call:
## lm(formula = Y99Pop ~ Y89Births + Y89Brate, data = mydat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -362649 -117800  -10240   36497  597511
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 502592.59  199219.41    2.52    0.015 *
## Y89Births       38.05       2.03   18.76   <2e-16 ***
## Y89Brate    -14369.09    5774.65   -2.49    0.017 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 215000 on 45 degrees of freedom
## Multiple R-squared: 0.898,Adjusted R-squared: 0.894
## F-statistic:  199 on 2 and 45 DF,  p-value: <2e-16
##


myregsum <- summary(myreg)  #create a new regression summary object

names(myregsum)

##  [1] "call"          "terms"         "residuals"     "coefficients"  "aliased"
##  [6] "sigma"         "df"            "r.squared"     "adj.r.squared" "fstatistic"
## [11] "cov.unscaled"


myregsum[["adj.r.squared"]]  #extract the adjusted r squared

## [1] 0.8938


myregsum$adj.r.squared  #does the same thing

## [1] 0.8938
```

That concludes our basic introduction to data.frames and lists. There is alot more material out on the web if you are interested. Later in the exercise we will look at data.frames in more detail.

1.4 Custom Functions

It is hard to unleash the full potential of R without writing your own functions. Luckily it's very easy to do. Here are some trivial examples:

```
# -------------CUSTOM FUNCTIONS-------------------

add <- function(x) {
    #put the function arguments in () and the evaluation in {}
    x + 1
}
add(3)
```

```
## [1] 4

add(4)

## [1] 5


# -Set the default values for your function-
add <- function(x = 5) {
    x + 1
}
add()  #automatically evaluates x=5

## [1] 6

add(6)  #but you can still change the defaults

## [1] 7
```

That's about all there is too it. The function will generally return the result of the last line that was evaluated. However you can also use `return()` to specify exactly what the function will return.

Functions can also return other functions. This concept is known as 'closures' and can be a very powerful tool. Here are some trivial examples (courtesy of H. Wickham's 'R Masters Class'):

```
# -------------FUN WITH CLOSURES------------------
power <- function(exponent) {
    function(x) {
        x^exponent
    }
}

square <- power(2)  #create a function called square
square(2)  #run the function and give it 2 as an argument

## [1] 4

square(4)

## [1] 16


cube <- power(3)  #create a function called cube
cube(2)

## [1] 8

cube(4)

## [1] 64
```

2 Set Your Working Directory and Load Your Libraries

2.1 Set the Working Directory

Let's do some basic set up first. In the code block below type in the file path to where your data is being held and then (if you want) use the setwd() (set working directory) command to give R a default location to look for data files.

```
# -------------BASIC SET UP---------------

# --Clear the workspace rm(list=ls()) #commented out for now, but a good way to start
# most R scripts

# --Set the working directory----
datdir <- "/Users/frankdavenport/Education/R_Work/SVN/broom/data/"  #This is an example
of a Mac file path

# datdir<-'H:/broom/data/' #This is an example of a PC file path (USE THIS IF YOU ARE ON
# A BROOM MACHINE)


# setwd(datdir) #This sets the working directory (where R looks for files)- NOT NECESSARY
# FOR THE BROOM CLASS
```

2.2 Load Libraries

Next we will load a series of R packages that will give the functions we need to complete all the exercises in this document. R 'packages' are user contributed functions. There are about 5000 or so (with a constantly expanding list). If a package is already installed you load the package with the `library()` command. If you want to install a package you can use the `install.packages()` command (you have to provide the url of the CRAN mirror to download the package from–see the R website for more details). If you are using R Studio you can also just click on Tools>Install Packages, and type in the name(s) of the package you want to install.

For this exercise all of the packages should (hopefully) be already installed on your machine. We will load them below using the library() command. I also included some comments describing how we use each of the packages in the exercises.

```
# -------------LOADING PACKAGES-------------------

# --Packages for Reading/Writing/Manipulating Spatial Data------
library(rgdal)  #conatins the read/writeOGR for reading shapelies and read/writeRGDAL for
reading raster data
library(maptools)  #Contains the overlay command
gpclibPermit()  #Makes all of the function in the maptools package available to us

## [1] TRUE


library(spdep)  #Contains a number of useful spatial stat functions
library(spatstat)  #Contains functions for generating random points drawn from a specific
data generating process

library(raster)  #contains a number of useful functions for raster data, especially
extract()
# ==================================================================

# --Packages for Data Visualization and Manipulation--
library(ggplot2)
library(reshape2)
library(scales)
# ====================================================
```

Note: Mention the importance of `gpclibPermit()` **Note:** Mention installing rgdal on a Mac,

3 Read and Plot Spatial Data

3.1 Read in a Shapefile

The most flexible way to read in a shapefile is by using the `readOGR` command. This is the only option that will also read in the .prj file associated with the shapefile. NCEAS has a useful summary of the various ways to read in a shapefile: `http://www.nceas.ucsb.edu/scicomp/usecases/ReadWriteESRIShapeFiles`

I recommend always using `readOGR()`.

Read OGR can be used for almost any vector data format. To read in a shapefile, you enter two arguments:

- dsn- The directory containing the shapefile (even if this is already your working directory)
- layer- the name of the shapefile, without the file extension

```
# -------------READ IN A SHAPEFILE-----------------

ds <- readOGR(dsn = datdir, layer = "kenya")

## OGR data source with driver: ESRI Shapefile
## Source: "/Users/frankdavenport/Education/R_Work/SVN/broom/data/", layer: "kenya"
## with 41 features and 2 fields
## Feature type: wkbPolygon with 2 dimensions
```

We can explore some basic aspects of the data using `summary()` and `str()`. Summary works on almost all R objects but returns different results depending on the type of object. For example if the object is the result of a linear regression then summary will give you the coefficient estimates, standard errors, t-stats, R^2, et cetera.

```
# -------------EXPLORE THE DATA-------------------
summary(ds)

## Object of class SpatialPolygonsDataFrame
## Coordinates:
##       min    max
## x 33.909 41.899
## y -4.678  4.629
## Is projected: FALSE
## proj4string : [+proj=longlat +ellps=clrk80 +no_defs]
## Data attributes:
##     ip89DId                ip89DName
##  Min.   :1010   Baringo        : 1
##  1st Qu.:3050   Bugoma         : 1
##  Median :5030   Busia          : 1
##  Mean   :5090   Elgeyo-Marakwet: 1
##  3rd Qu.:7060   Embu           : 1
##  Max.   :8030   Garissa        : 1
##                 (Other)        :35


str(ds, 2)
```

```
## Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
##   ..@ data       :'data.frame': 41 obs. of  2 variables:
##   ..@ polygons   :List of 41
##   ..@ plotOrder  : int [1:41] 17 36 21 19 12 15 20 14 26 34 ...
##   ..@ bbox       : num [1:2, 1:2] 33.91 -4.68 41.9 4.63
##   .. ..- attr(*, "dimnames")=List of 2
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
```

As mentioned above, the `summary()` command works on virtually all R objects. In this case it gives some basic information about the projection, coordinates, and data contained in our shapefile

The `str()` or structure command tells us how R is actually storing and organizing our shapefile. This is a useful way to explore complex objects in R. When we use `str()` on a spatial polygon object, it tells us the object has five 'slots':

1. *data*: This holds the data.frame
2. *polygons*: This holds the coordinates of the polygons
3. *plotOrder*: The order that the coordinates should be drawn
4. *bbox*: The coordinates of the bounding box (edges of the shape file)
5. *proj4string*: A character string describing the projection system

The only one we want to worry about is data, because this is where the data.frame() associated with our spatial object is stored. We access slots using the @ sign.

Note Mention S3 vs S4 classes?

```
# -------------ACCESS THE SHAPEFILE DATA---------------
dsdat <- ds@data  #extract the data into a regular data.frame
head(dsdat)

##   ip89DId ip89DName
## 0    1010   Nairobi
## 1    2010    Kiambu
## 2    2020 Kirinyaga
## 3    2030   Muranga
## 4    2040 Nyandaura
## 5    2050     Nyeri


ds@data$new <- 1:nrow(dsdat)  #add a new column, just like adding data to a data.frame
head(ds@data)

##   ip89DId ip89DName new
## 0    1010   Nairobi   1
## 1    2010    Kiambu   2
## 2    2020 Kirinyaga   3
## 3    2030   Muranga   4
## 4    2040 Nyandaura   5
## 5    2050     Nyeri   6
```

3.2 Plotting the Data

Plotting is easy, use the `plot()` command:

```
# -------------PLOT THE SHAPEFILE------------------

plot(ds)
```

Obviously there are more options to dress up your plot and make a proper map/graphic. A common method is to use `spplot()` from the sp package. However I prefer to use the functions available in the ggplot2 package as I think they are more flexible and intuitive. We will address maps and graphics later in the in the class. For now, let us move onto reading in some tabular data and merging that data to our shapefile (similar to the join operation in ArcGIS).

4 Read in a .csv File and Join it to the Shapefile

4.1 Read in a .csv file

First lets read in a *.csv* file using `read.csv()`

```
# -------------READ AND EXPLORE A CSV---------------

d <- read.csv(paste(datdir, "kenpop89to99.csv", sep = ""))
# Use summary()get a quick look at the data:
summary(d)

##     ip89DId          ip89DName         ADMIN3     KEADMN3_ID        Y89Pop
##  Min.   :1010   Kisii        : 3   KISII     : 2   Min.   : 1.0   Min.   :  57960
##  1st Qu.:3772   Kakamega     : 2   BARINGO   : 1   1st Qu.:12.8   1st Qu.: 222905
##  Median :6010   Kericho      : 2   BOMET     : 1   Median :24.5   Median : 451510
##  Mean   :5207   Machakos     : 2   BUNGOMA   : 1   Mean   :25.5   Mean   : 619710
##  3rd Qu.:7052   Meru         : 2   BUSIA     : 1   3rd Qu.:35.2   3rd Qu.: 947500
##  Max.   :8030   South Nyanza: 2   E. MARAKWET: 1   Max.   :63.0   Max.   :1476500
```

```
##                  (Other)     :35   (Other)    :41
##    Y89Births        Y89Brate        Y99Pop          Y99Births        Y99Brate
##   Min.   : 1680   Min.   :22.6   Min.   :  72380   Min.   : 1760   Min.   :19.0
##   1st Qu.: 9350   1st Qu.:33.5   1st Qu.: 392545   1st Qu.:10870   1st Qu.:28.0
##   Median :18270   Median :37.4   Median : 629740   Median :21820   Median :31.0
##   Mean   :23719   Mean   :37.0   Mean   : 872928   Mean   :27562   Mean   :31.6
##   3rd Qu.:39855   3rd Qu.:40.9   3rd Qu.:1384665   3rd Qu.:42140   3rd Qu.:36.4
##   Max.   :57460   Max.   :51.0   Max.   :2363120   Max.   :69380   Max.   :42.9
##
##      PopChg          BrateChg
##   Min.   :-14.0   Min.   :-38.00
##   1st Qu.: 23.8   1st Qu.:-20.00
##   Median : 33.5   Median :-14.00
##   Mean   : 47.7   Mean   :-14.56
##   3rd Qu.: 44.2   3rd Qu.: -6.75
##   Max.   :343.0   Max.   :  0.00
##

# head(d) #first six rows tail(d) #last six rows

# -If you are using RStudio-Click on the Workspace Tab, then click on 'd' and you will
# get a spreadsheet view of the data. If you are not using RStudio you can get the same
# result by typing fix(d)
```

Before we merge the csv file to our shapefile, let's do some basic cleaning. The csv file has some excess columns and rows. Let's get rid of them. We access rows and columns by using square brackets [,].

Here are some examples using are data.frame 'd':

- d[1,] first row, all columns
- d[,1] first column all rows
- d[1,1] item in the first row and first column
- d[,1:5] columns 1 through 5 (also works with rows)
- d[,c(1,4,5)] columns 1,4 and 5 (also works with rows)
- d[,'variable'] column names 'variable'
- d$variable same as above, but returns the column as a vector
- d[d$variable>10,] rows from all columns that correspond where the values in 'variable' are greater than 10

Hopefully you get the idea. See the R cheat sheet: `http://cran.r-project.org/doc/contrib/Short-refcard.pdf` for more information.

Now we extract only the columns we we want and then use the unique() command to get rid of duplicate rows.

```
# -------------EXTRACT COLUMNS FROM CSV---------------

d <- d[, c("ip89DId", "PopChg", "BrateChg", "Y89Pop", "Y99Pop")]  #Grab only the columns
we want
summary(d)
```

```
##      ip89DId          PopChg          BrateChg          Y89Pop            Y99Pop
##   Min.   :1010   Min.   :-14.0   Min.   :-38.00   Min.   :  57960   Min.   :  72380
##   1st Qu.:3772   1st Qu.: 23.8   1st Qu.:-20.00   1st Qu.: 222905   1st Qu.: 392545
##   Median :6010   Median : 33.5   Median :-14.00   Median : 451510   Median : 629740
##   Mean   :5207   Mean   : 47.7   Mean   :-14.56   Mean   : 619710   Mean   : 872928
##   3rd Qu.:7052   3rd Qu.: 44.2   3rd Qu.: -6.75   3rd Qu.: 947500   3rd Qu.:1384665
##   Max.   :8030   Max.   :343.0   Max.   :  0.00   Max.   :1476500   Max.   :2363120

nrow(d)

## [1] 48

d <- unique(d)  #get rid of duplicate rows
nrow(d)  #note we now have less rows

## [1] 41
```

4.2 Join the csv file to our Shapefile

In R there a variety of options available for joining data sets. The most simple and intuitive is the `merge()` command (see ?merge for details). Merge takes two data.frames and matches them based on common attributes in columns. If the user does not specify the name(s) of the columns then merge will just look for common column names and perform the join on those. However with spatial objects the process is a little more tricky. Unfortunately merge automatically re-orders the new merged data.frame based on the common columns. This will not work for a spatial object as the associated shapes (points, lines, or polygons) would have to be reordered as well. There are a variety of ways around this and I will show a simple one below.

First I will demonstrate the basic merge() function. Then I will show one method for merging tabular to spatial data.

```
# --------EXPLORE MERGE AND DO A TABLE JOIN---------------

# ----------First a basic Merge Just to Demonstrate-------
d2 <- ds@data  #Extract the data

d3 <- merge(d, d2)  #They have common column names so we don't have to specify what to
join on
head(d3)

##   ip89DId PopChg BrateChg  Y89Pop  Y99Pop ip89DName new
## 1    1010     57      -12 1325620 2085820   Nairobi   1
## 2    2010     52      -14  908120 1383300    Kiambu   2
## 3    2020     16      -15  389440  452180 Kirinyaga   3
## 4    2030    -14      -31  862540  737520   Muranga   4
## 5    2040     34      -21  348520  468300 Nyandaura   5
## 6    2050      6      -23  607980  644380     Nyeri   6


d4 <- merge(ds, d)  #This will produce the same result.
head(d4)

##   ip89DId ip89DName new PopChg BrateChg  Y89Pop  Y99Pop
## 1    1010   Nairobi   1     57      -12 1325620 2085820
## 2    2010    Kiambu   2     52      -14  908120 1383300
```

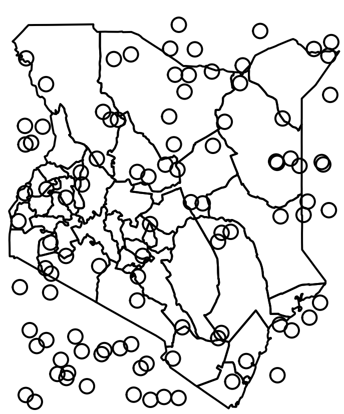
```
## 3    2020 Kirinyaga   3     16      -15  389440  452180
## 4    2030   Muranga   4    -14      -31  862540  737520
## 5    2040 Nyandaura   5     34      -21  348520  468300
## 6    2050     Nyeri   6      6      -23  607980  644380

# =====================================================

# -------Now lets do the Table Join: Join csv data to our Shapefile---

# -We can do the join in one line by using the match() function
ds@data <- data.frame(ds@data, d[match(ds@data[, "ip89DId"], d[, "ip89DId"]), ])

summary(ds)

## Object of class SpatialPolygonsDataFrame
## Coordinates:
##      min    max
## x 33.909 41.899
## y -4.678  4.629
## Is projected: FALSE
## proj4string : [+proj=longlat +ellps=clrk80 +no_defs]
## Data attributes:
##     ip89DId               ip89DName       new       ip89DId.1        PopChg
##  Min.   :1010   Baringo        : 1   Min.   : 1   Min.   :1010   Min.   :-14.0
##  1st Qu.:3050   Bugoma         : 1   1st Qu.:11   1st Qu.:3050   1st Qu.: 24.0
##  Median :5030   Busia          : 1   Median :21   Median :5030   Median : 34.0
##  Mean   :5090   Elgeyo-Marakwet: 1   Mean   :21   Mean   :5090   Mean   : 47.3
##  3rd Qu.:7060   Embu           : 1   3rd Qu.:31   3rd Qu.:7060   3rd Qu.: 42.0
##  Max.   :8030   Garissa        : 1   Max.   :41   Max.   :8030   Max.   :343.0
##                 (Other)        :35
##     BrateChg          Y89Pop             Y99Pop
##  Min.   :-38.0   Min.   :  57960   Min.   :  72380
##  1st Qu.:-18.0   1st Qu.: 214240   1st Qu.: 324180
##  Median :-12.0   Median : 403760   Median : 547680
##  Mean   :-13.4   Mean   : 523950   Mean   : 721160
##  3rd Qu.: -6.0   3rd Qu.: 741940   3rd Qu.: 813200
##  Max.   :  0.0   Max.   :1476500   Max.   :2363120
##


# head(ds@data)
# ===========================================================================
```

Note that the values from our csv are not in the data attributes of the shapefile. Note also that we have duplicated the join field 'ip89DId'. We can delete it afterwards but it's a nice way to double check and make sure our join worked correctly. I will go over the details of this approach in class and you can also see an explanation here:
`http://stackoverflow.com/questions/3650636/how-to-attach-a-simple-data-frame-to-a-spatialpolygondatafra`
`in-r`

5 Create Random Points and Extract as a Text File

Just like in the ArcGIS lab that preceded this one, we are going to do a point in polygon spatial join. However before we do that we are going to generate some random points. We will use the function `runifpoint()` from the spatstat package. This function creates N points drawn from a spatial uniform distribution (complete spatial randomness) within a given bouning box. The bounding box can be in a variety of forms but the most straightforward is simply a four element vector with *xmin* (the minimum x coordinate), *xmax*, *ymin*, and *ymax*. In the code below we will extract this box from our Kenya data set, convert it to a vector, generate the points, and then plot the points on top of the Kenya map.

```
# -------------GENERATE RANDOM POINTS---------------

win <- bbox(ds)  #the bounding box around the Kenya dataset
win

##      min    max
## x 33.909 41.899
## y -4.678  4.629

win <- t(win)  #transpose the bounding box matrix
win

##         x      y
## min 33.91 -4.678
## max 41.90  4.629

win <- as.vector(win)  #convert to a vector for input into runifpoint()
win

## [1] 33.909 41.899 -4.678  4.629

dran <- runifpoint(100, win = as.vector(t(bbox(ds))))  #create 100 random points

plot(ds)
plot(dran, add = T)
```



Now that we have created some random points, we will extract the x coordinates (longitude), y coordinates (latitude), and then simulate some values to go with them. The purpose of doing this is to create a file similar to the the random points file we used in the ArcGIS exercise: A text file with x,y, and some values. We will then write those values out as a .csv file, read them back in, convert them to a shapefile, and then do a point in polygon spatial join.

```
# -------------CONVERT RANDOM POINTS TO DATA.FRAME---------
dp <- as.data.frame(dran)  #This creates a simple data frame with 2 colunms, x and y
head(dp)

##       x      y
## 1 37.78 2.3948
## 2 41.65 0.1244
## 3 38.41 4.0228
## 4 34.32 3.8055
## 5 39.56 3.6338
## 6 34.72 2.1490


# Now we will add some values that will be aggregated in the next exercise
dp$values <- rnorm(100, 5, 10)  #generates 100 values from a Normal distribution with
mean 5, and sd-10
head(dp)

##       x      y values
## 1 37.78 2.3948 23.710
## 2 41.65 0.1244  4.545
## 3 38.41 4.0228 14.542
## 4 34.32 3.8055  3.960
## 5 39.56 3.6338  8.033
## 6 34.72 2.1490  8.913
```

6 Do a Point in Polygon Spatial Join

In the last exercise we generated some random points along with some random values. Now we will read that data in, convert it to a shapefile (or a SpatialPointsDataFrame object) and then do a point in polygon spatial join. The command for converting coordinates to spatial points is `SpatialPointsDataFrame()`

```
# -------------CONVERT RANDOM POINTS TO SPATIAL POINTS DATAFRAME--
dsp <- SpatialPointsDataFrame(coords = dp[, c("x", "y")], data = data.frame(values =
dp$values))
summary(dsp)

## Object of class SpatialPointsDataFrame
## Coordinates:
##     min    max
## x 34.14 41.706
## y -4.52  4.618
## Is projected: NA
## proj4string : [NA]
## Number of points: 100
## Data attributes:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -24.300  -0.455   5.780   5.890  12.700  28.100
```

```r
# --Since the Data was Generated from a source with same projection as our Kenya data,
# we will go head and define the projection'

dsp@proj4string <- ds@proj4string
```

Now that we have created some points and defined their projection, we are ready to do a point in polygon spatial join. We will use the `over()` command (short for overlay()).

In the over() command we feed it a spatial polygon object (ds), a spatial points object (dsp), and tell it what function we want to use to aggregate the spatial point up. In this case we will use the mean (but we could use any function or write our own). The result will give us a data.frame, and we will then put the resulting aggregated values back into the data.frame() associated with ds (ds@data).

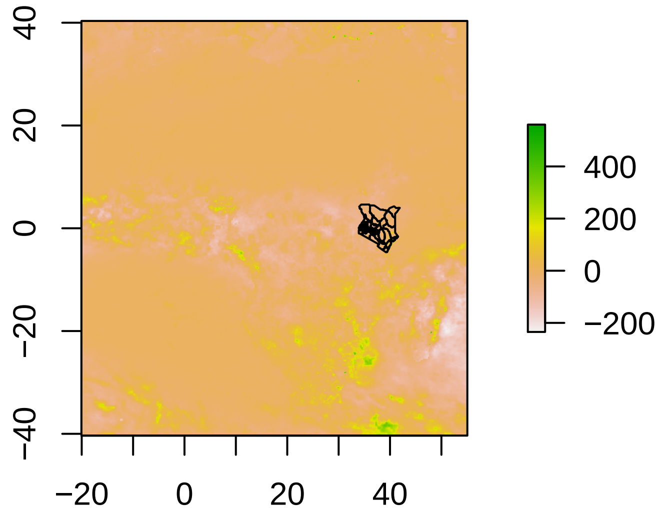
See ?over() for more information.

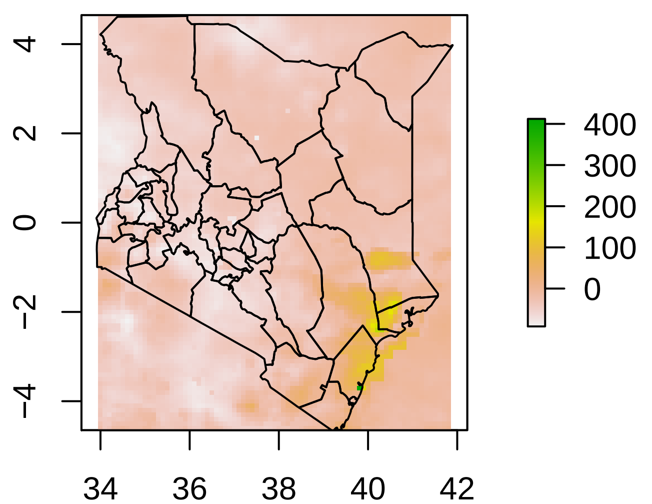
```r
# -------------POINT IN POLY JOIN---------------

# -The data frame tells us for each point the index of the polygon it falls into
dsdat <- over(ds, dsp, fn = mean)  #do the join
head(dsdat)  #look at the data

##   values
## 0     NA
## 1     NA
## 2 -11.27
## 3  16.17
## 4     NA
## 5     NA


inds <- row.names(dsdat)  #get the row names of dsdat so that we can put the data back
into ds
head(inds)

## [1] "0" "1" "2" "3" "4" "5"


ds@data[inds, "pntvals"] <- dsdat  #use the row names from dsdata to add the aggregated
point values to ds@data
head(ds@data)

##   ip89DId ip89DName new ip89DId.1 PopChg BrateChg   Y89Pop   Y99Pop pntvals
## 0    1010   Nairobi   1      1010     57      -12 1325620 2085820      NA
## 1    2010    Kiambu   2      2010     52      -14  908120 1383300      NA
## 2    2020 Kirinyaga   3      2020     16      -15  389440  452180  -11.27
## 3    2030   Muranga   4      2030    -14      -31  862540  737520   16.17
## 4    2040 Nyandaura   5      2040     34      -21  348520  468300      NA
## 5    2050     Nyeri   6      2050      6      -23  607980  644380      NA
```

7 Do a Pixel in Polygon Spatial Join

In this section we will explore another common spatial join operation. In this case you you have raster data that you want to aggregate up to the level of the polygons. A common example is that you have a surface of observed or interpolated temperature measurements and you want to find out what the average (or sum, max, min, et cetera) temperature is for each polygon (which could represent states, counties, et cetera).

```
# -------------READ AND CROP A RASTER---------------

g <- readGDAL(fname = "data/anom.2000.03.tiff")  #Read in a geoTiff of rainfall anomolies

## data/anom.2000.03.tiff has GDAL driver GTiff
## and has 801 rows and 751 columns

g <- raster(g)  #convert it to a format recongnizable by the raster package

# -plot it
plot(g)
plot(ds, add = T)  #plot kenay on top to get some sense of the extent

# ---Crop the Raster Dataset to the Extent of the Kenya Shapefile
gc <- crop(g, ds)  #clip the raster to the extent of the shapefile

# Then test again to make sure they line up
plot(gc)
plot(ds, add = T)
```





In the last step we read in a raster file, cropped it to the extent of the Kenya data (just to cut down on the file size and demonstrate that function). Now we will aggregate the pixel values up the polygon values using the `extract()` function.

```
# -------------PIXEL IN POLY SPATIAL JOIN---------------

# Unweighted- only assigns grid to district if centroid is in that district
ds@data$precip <- extract(gc, ds, fun = mean, weights = FALSE)

## Warning message: Transforming SpatialPolygons to the CRS of the Raster


# Weighted (more accurate, but slower)- weights aggregation by the amount of the grid
# cell that falls within the district boundary
# ds@data$precip_wght<-extract(gc,ds,fun=mean,weights=TRUE)

# -If you want to see the actual values and the weights associated with them do this:
# rastweight<-extract(gc,ds,weights=TRUE)

# ==========================================================

# ---Examine the Results and Extract the Data------ Plot The Results
# spplot(dsp[,c('wrsi','wrsi_wght')])
```

Now that we've added all this data to our shapefile, we'll write it out as a new shapefile and then load it in to make some maps in the next exercise.

8 Make Maps with ggplot2()

If you have not already done so, load ggplot2 and some related packages.

For more info on the ggplot2 and the grammar of graphics see the resources at http://had.co.nz/ggplot2/.

The 'gg' in the ggplot2 is short for *The Grammar of Graphics* which references a famous book by the same name. The idea behind the book and the software is to try and decompose any graphic into a set of fundamental elements. We can then use these elements to construct any type of graphic we want (the elements are the grammar), rather than having a different command for every type of graphic out there. We do not have time to do a full overview of ggplot2 but if you click on the link above and scroll down there is a good visual overview of how ggplot2 works. If you have time take a minute to visit the website.

8.1 Setting up the Data with fortify()

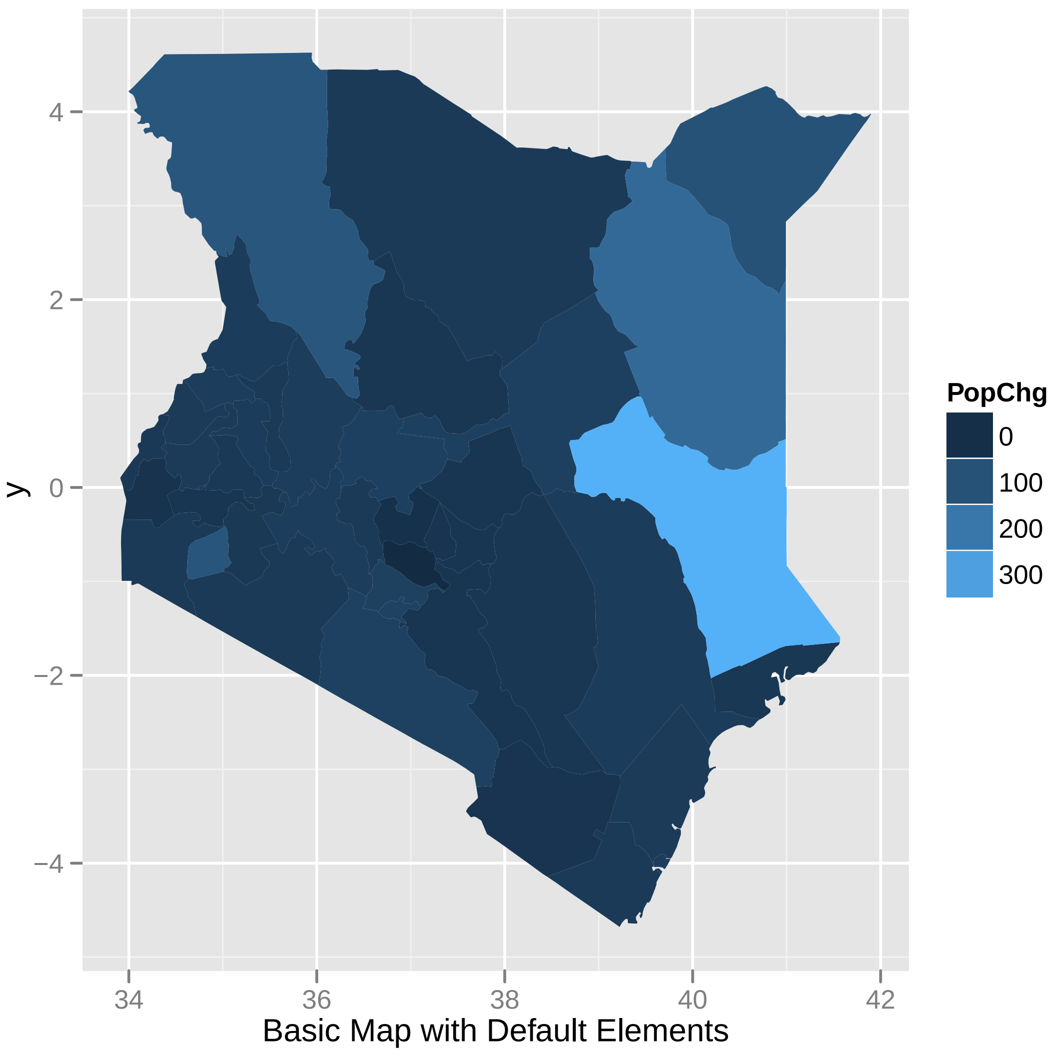
The ggplot2() package separates spatial data into 2 elements: 1) The data.frame and 2)The spatial coordinates. If you want to make a map from a shapefile you first have to use the fortify() command which converts the shapefile to a format readable by ggplot2:

```
# ------------PREP SPATIAL DATA FOR GGPLOT WITH FORTIFY()------

pds <- fortify(ds)  #convert to form readable by ggplot2

## Using ip89DId to define regions.

pds$ip89DId <- as.integer(pds$id)
head(pds)

##    long    lat order  hole piece  group   id ip89DId
## 1 36.91 -1.165     1 FALSE     1 1010.1 1010    1010
## 2 36.91 -1.165     2 FALSE     1 1010.1 1010    1010
## 3 36.92 -1.165     3 FALSE     1 1010.1 1010    1010
```

```
## 4 36.94 -1.176      4 FALSE      1 1010.1 1010     1010
## 5 36.94 -1.179      5 FALSE      1 1010.1 1010     1010
## 6 36.94 -1.181      6 FALSE      1 1010.1 1010     1010
```

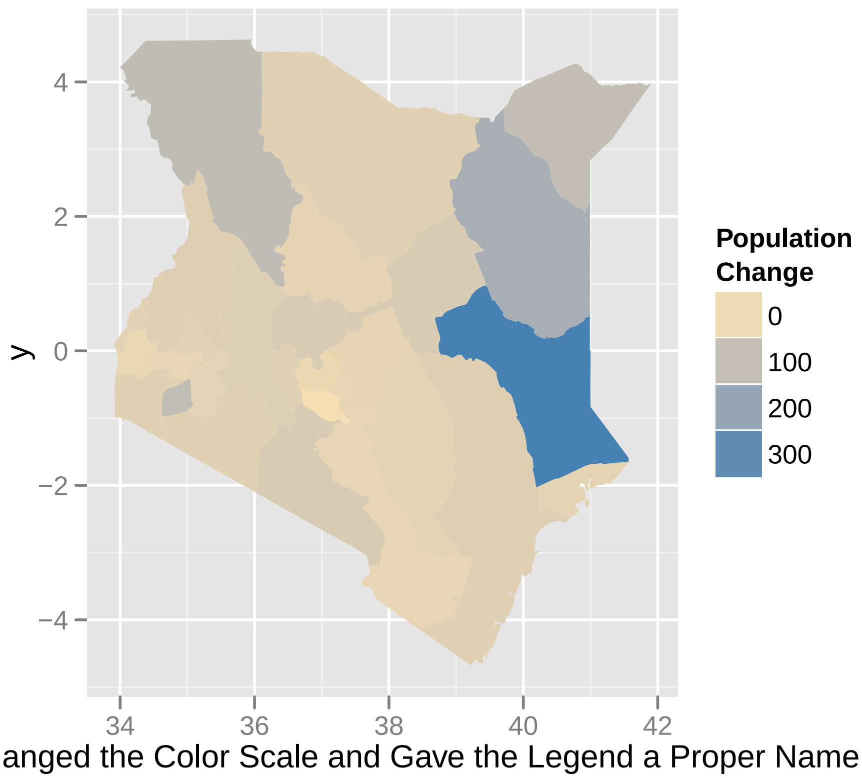
Now, we will build the map step by step using ggplot2. We could do it all in one line, but it's easier to do it one step at a time so you can see how the different elements combine to make the final graphic. In the code below we will first create the basic layer using the ggplot command, and then we customize to it.

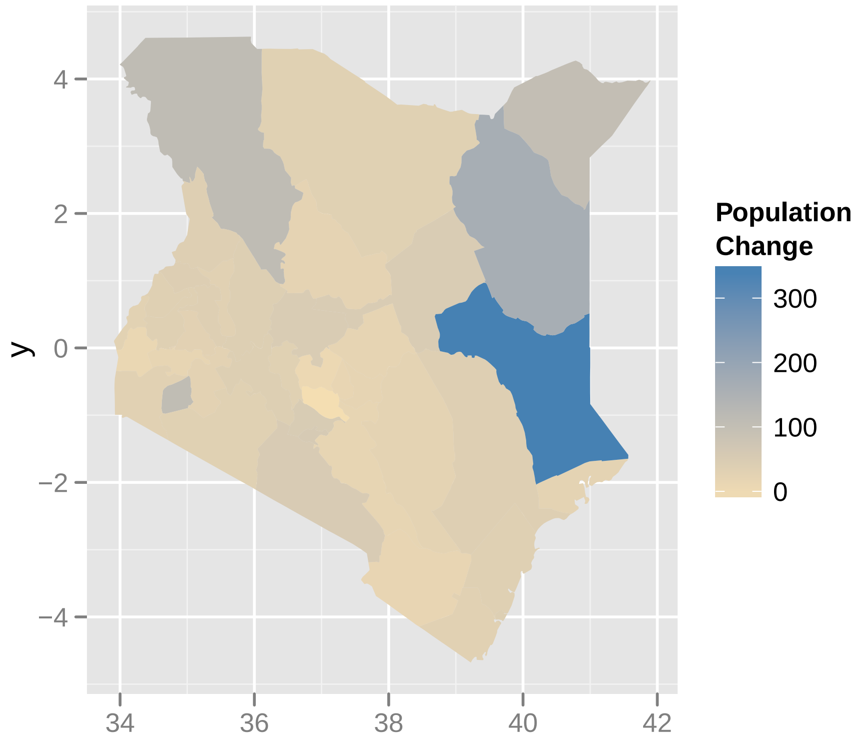
```
# -------------MAKE A BASIC MAP---------------


# Make the Map
p1 <- ggplot(d, aes(map_id = ip89DId))
p1 <- p1 + geom_map(aes(fill = PopChg, map_id = ip89DId), map = pds)
p1 <- p1 + expand_limits(x = pds$lon, y = pds$lat) + coord_equal()
p1 + xlab("Basic Map with Default Elements")
```



Now we have a basic map, let's make some tweaks to it.

```
# -------------CHANGE THE COLOR SCHEME, TWEAK THE LEGEND------


# --Change the Colour Scheme---
p1 <- p1 + scale_fill_gradient(name = "Population \nChange", low = "wheat", high =
"steelblue")  #to set break points, enter in breaks=c(...,..)
# The \n in Population \nChange' indicates a carriage return
p1 + xlab("We Changed the Color Scale and Gave the Legend a Proper Name")

# ---Tweak the Legend--
p1 <- p1 + guides(fill = "colorbar")  #for more advanced colorabr options see
?guide_colorbar()
p1 + xlab("Now the Legend is a Colorbar \n which better Represents Continuous Data")
```

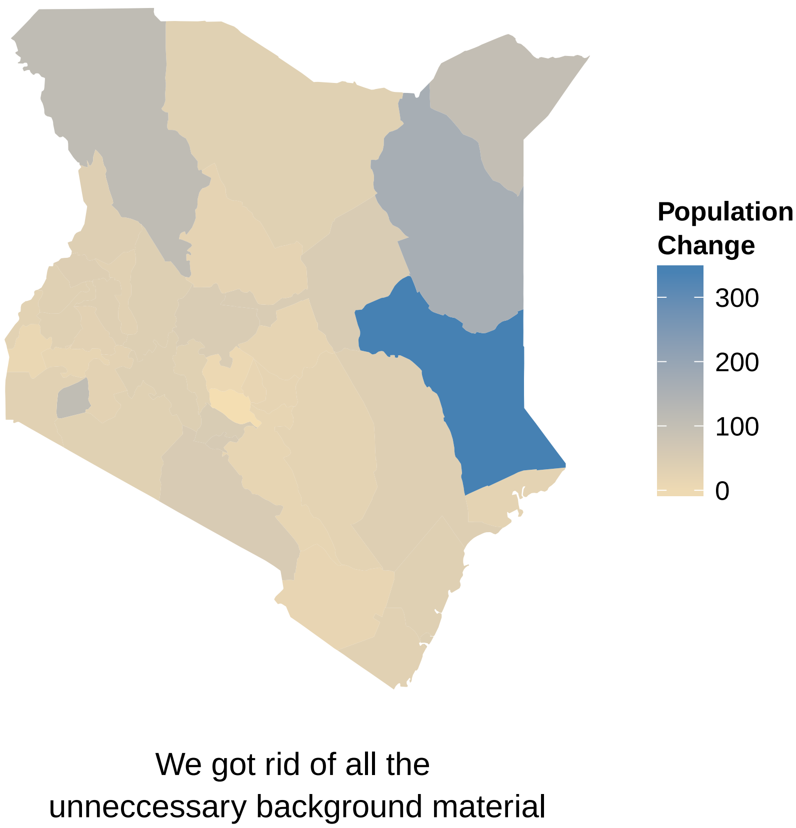




Now the Legend is a Colorbar
which better Represents Continuous Data

Now we will get rid of all the unnecessary information in the background.

```
# -------------EDIT THE BACKGROUND---------------

# ---Get Rid of the Background---

# Blank Grid, Background,Axis,and Tic Marks
bGrid <- opts(panel.grid.major = theme_blank(), panel.grid.minor = theme_blank())
bBack <- opts(panel.background = theme_blank())
bAxis <- opts(axis.title.y = theme_blank())
bTics <- opts(axis.text.x = theme_blank(), axis.text.y = theme_blank(), axis.ticks =
theme_blank())


p1 <- p1 + bAxis + bTics + bGrid + bBack
p1 + xlab("We got rid of all the \nunneccessary background material")
```



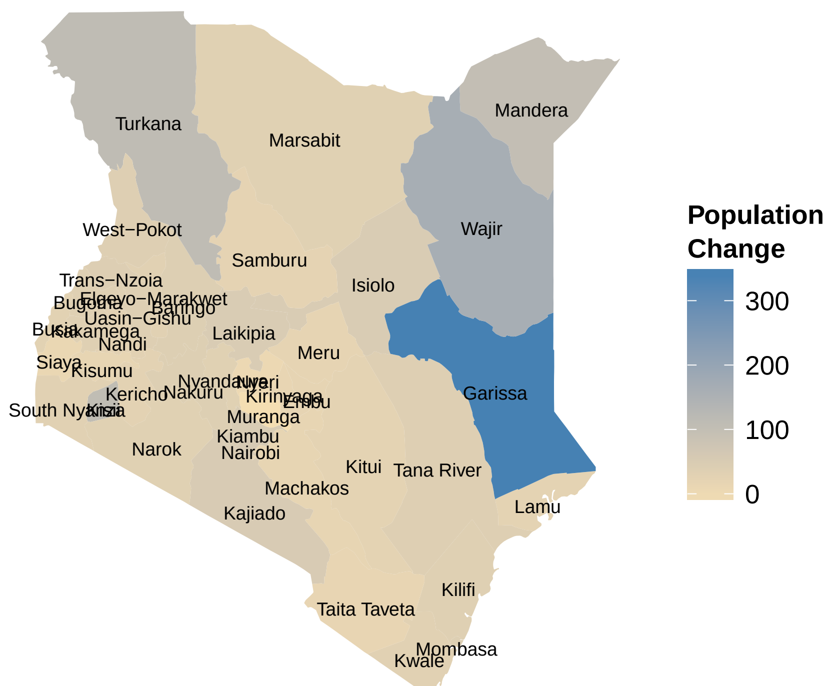
Now let's label the polygon names and data values.

```
# -------------ADD SOME LABELS-------------------


# ----Add Some Polygon labels--- -Polygon Labels
cens <- as.data.frame(coordinates(ds))  #extract the coordinates for centroid of each
polygon
cens$Region <- ds$ip89DName
cens$ip89DId <- ds$ip89DId
head(cens)  #we will use this file to label the polygons

##      V1      V2    Region ip89DId
## 1 36.86 -1.2985   Nairobi    1010
## 2 36.82 -1.0744    Kiambu    2010
## 3 37.32 -0.5266 Kirinyaga    2020
## 4 37.03 -0.8108   Muranga    2030
## 5 36.48 -0.3225 Nyandaura    2040
## 6 36.95 -0.3396     Nyeri    2050



p1 <- p1 + geom_text(data = cens, aes(V1, V2, label = Region), size = 2.5, vjust = 1)
p1 + xlab("We added some text labels \nfor the Various Spatial Units")
```

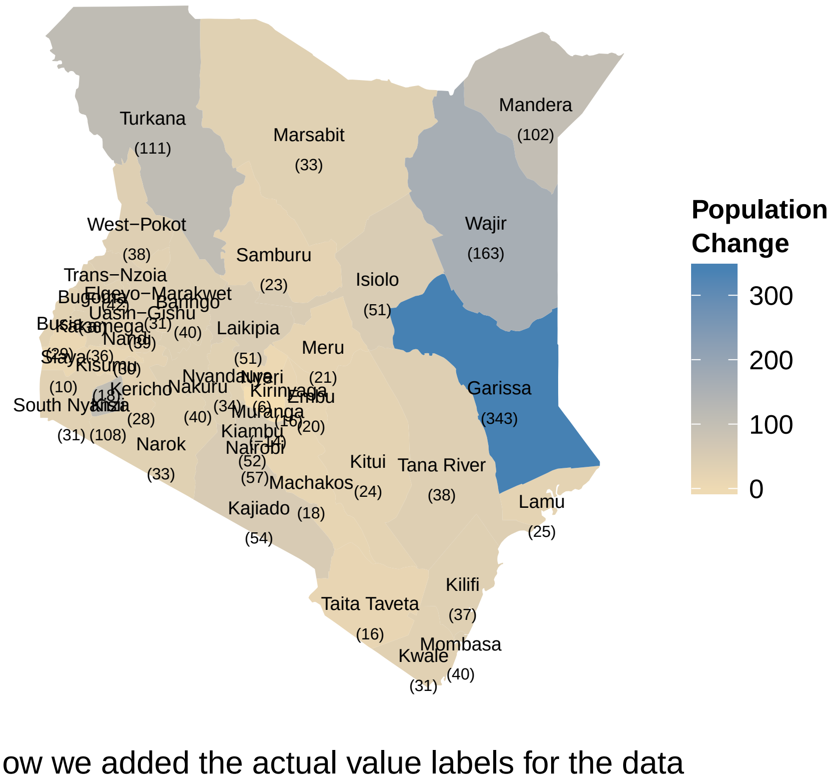
```r
# ---Add Some value Labels------
pdlab <- merge(cens, d)  #Merge the centroids with out data
head(pdlab)  #We will use this to label the polygons with their data values
```

```
##   ip89DId    V1      V2    Region PopChg BrateChg  Y89Pop  Y99Pop
## 1    1010 36.86 -1.2985   Nairobi     57      -12 1325620 2085820
## 2    2010 36.82 -1.0744    Kiambu     52      -14  908120 1383300
## 3    2020 37.32 -0.5266 Kirinyaga     16      -15  389440  452180
## 4    2030 37.03 -0.8108   Muranga    -14      -31  862540  737520
## 5    2040 36.48 -0.3225 Nyandaura     34      -21  348520  468300
## 6    2050 36.95 -0.3396     Nyeri      6      -23  607980  644380
```

```r
p1 <- p1 + geom_text(data = pdlab, aes(V1, V2, label = paste("(", PopChg, ")", sep =
"")),
    colour = "black", size = 2, vjust = 3.7)
p1 + xlab("Now we added the actual value labels for the data")
```

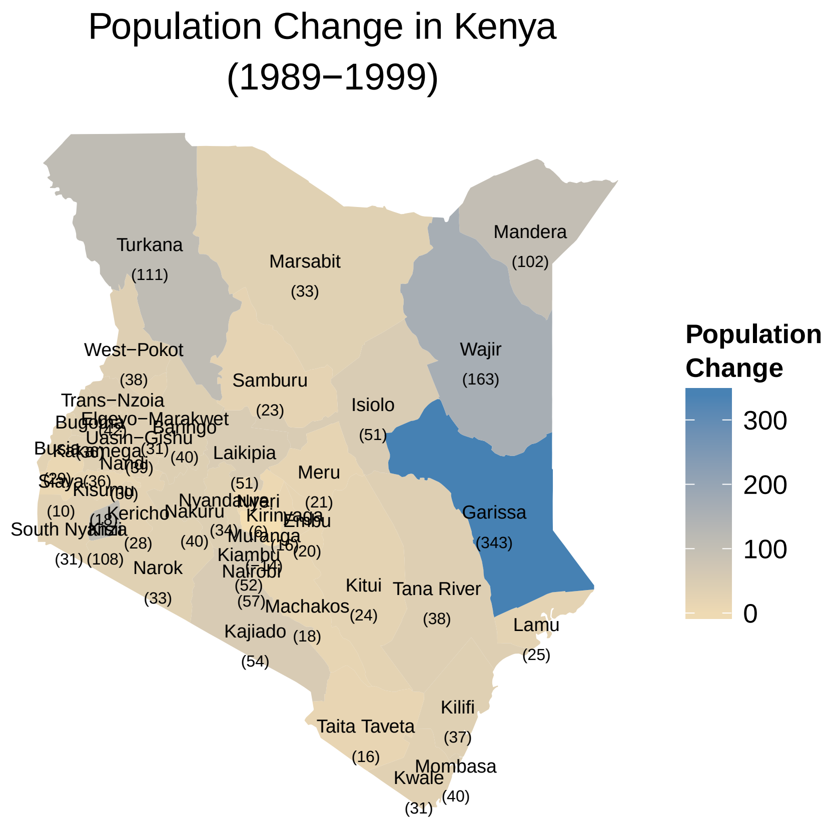


We added some text labels
for the Various Spatial Units



Finally we will add a title.

```
# ----Add a title---------
p1 <- p1 + opts(title = "Population Change in Kenya \n (1989-1999)")
p1 + xlab("Finally we add a title")
```



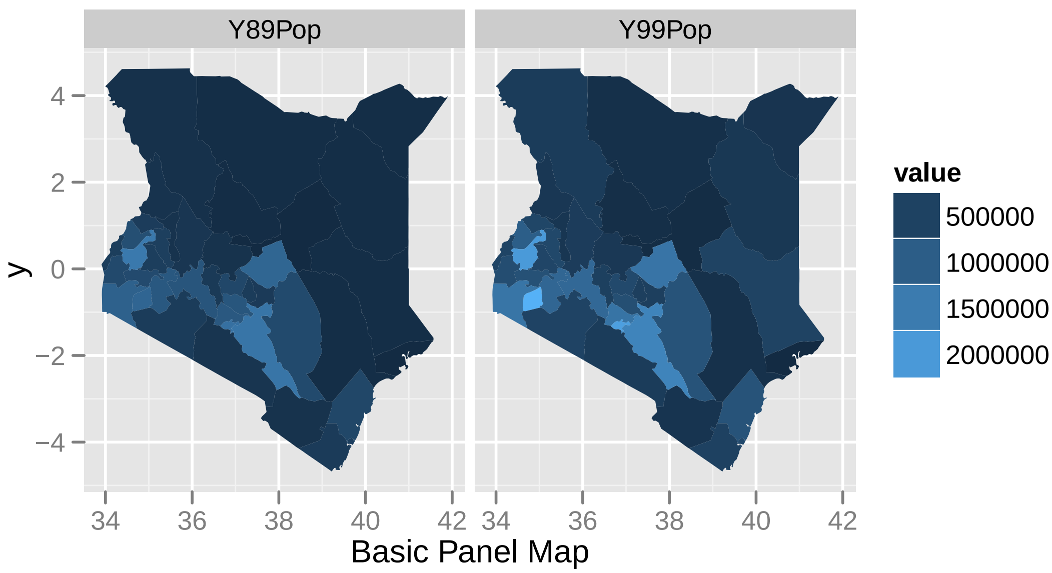
Finally we add a title

8.2 Plotting Panel Maps

So now we have made a basic map with a legend, location labels, and value labels. One of the advantages of ggplot is the ease with which you can create panel graphics, or to use the ggplot terminology 'faceting'. Imagine for example that you have a spatial panel data set- multiple observations of the same spatial feature over several years. Ggplot gives you several options for displaying this data using either the `facet_wrap()` or `facet_grid()` commands. In the example below we will make panel maps for the population data in the Kenya data set.

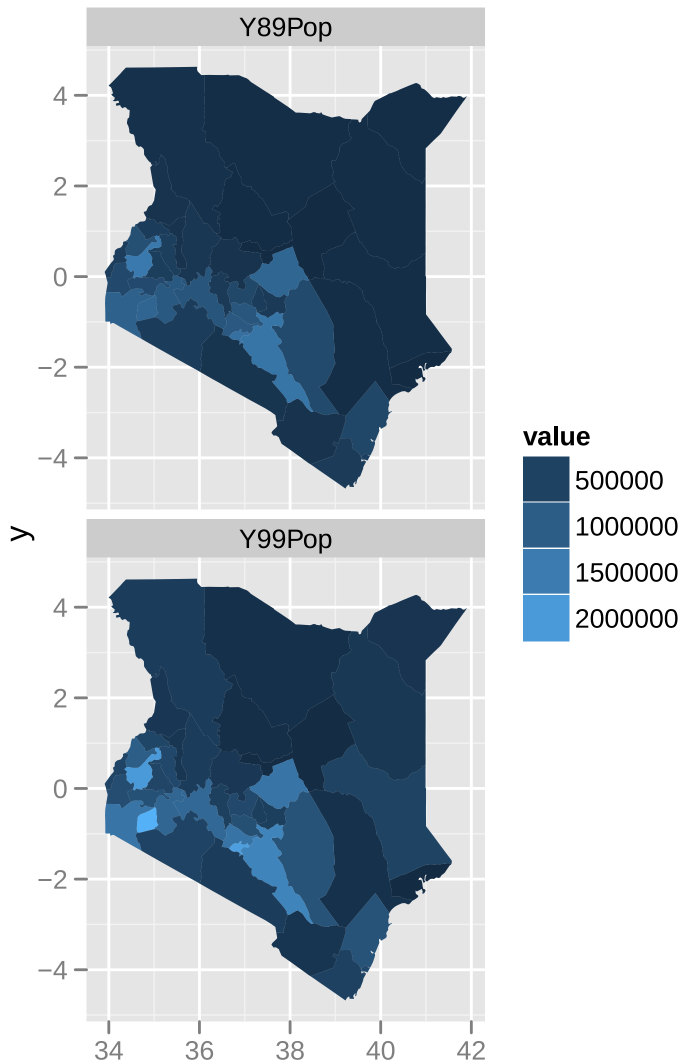
```
# -------------RESHAPE THE DATA AND MAKE A PANEL MAP--------


pd <- d[, c("ip89DId", "Y89Pop", "Y99Pop")]  #select out certain colunms
pd <- melt(pd, id.vars = "ip89DId")  #convert the data to 'long' form
head(pd)  #take a look at the data

##   ip89DId variable   value
## 1    1010   Y89Pop 1325620
## 2    2010   Y89Pop  908120
## 3    2020   Y89Pop  389440
## 4    2030   Y89Pop  862540
## 5    2040   Y89Pop  348520
## 6    2050   Y89Pop  607980
```

```
pmap <- ggplot(pd, aes(map_id = ip89DId))
p2 <- pmap + geom_map(aes(fill = value, map_id = ip89DId), map = pds) +
facet_wrap(~variable)
p2 <- p2 + expand_limits(x = pds$lon, y = pds$lat) + coord_equal()
p2 + xlab("Basic Panel Map")
```



We can use the `ncols' (number of columns) argument in facet_wrap() to make the panels stack vertically instead of horizontally.

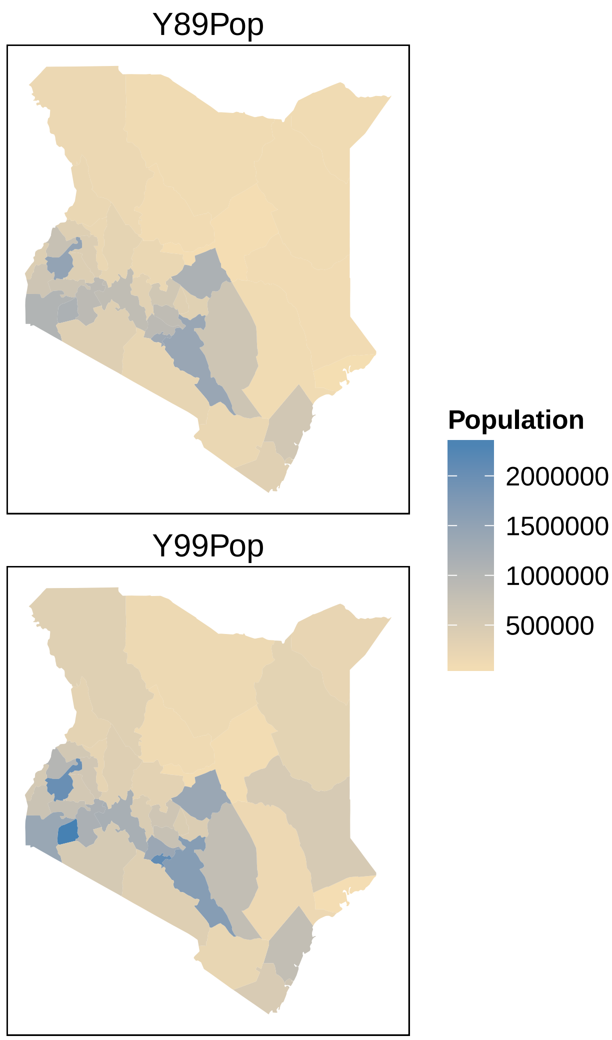
```
# -------------TWEAK THE PANEL MAP-----------------


# If we want to stack the panels vertically we change the options in facet_wrap()
p2 <- p2 + facet_wrap(~variable, ncol = 1)  #have only 1 column of panels
p2 + xlab("We change the option in facet_wrap so the panels are stacked")
```



We change the option in facet_wrap so the panels are stacked

Finally we can use the same options we used above to make our final map.

```
# -------------MORE PANEL MAP TWEAKS---------------


# -We can add all the other tweaks as before
p2 <- p2 + scale_fill_gradient(name = "Population", low = "wheat", high = "steelblue")
#to set break points, enter in
p2 <- p2 + guides(fill = "colorbar")
p2 <- p2 + bAxis + bGrid + bTics + bBack + opts(panel.border = theme_rect())  #this
removes the background but keeps a border around the panels

# -We can also adjust the format, theme, et cetera of the panel lables with
# 'strip.text.x'
p2 <- p2 + opts(strip.background = theme_blank(), strip.text.x = theme_text(size = 12))
p2 + xlab("Our Final Map")
# ==============================================================================
```



Our Final Map

That's it.