Take-Home Final Exam

Statistical Computing, 36-350

Due December 7, 2015 at 11:59pm

Bonus: Maps for the masses

For this bonus problem, you should use the cleaned data set from question 1 of the final exam. This cleaned `census` data frame should have 70877 rows and 31 columns. For simplicity, we've posted this cleaned census data to the class website, and below we load in the `plyr` package (*hint*: this will again be very useful for this bonus question), and read in the cleaned data frame.

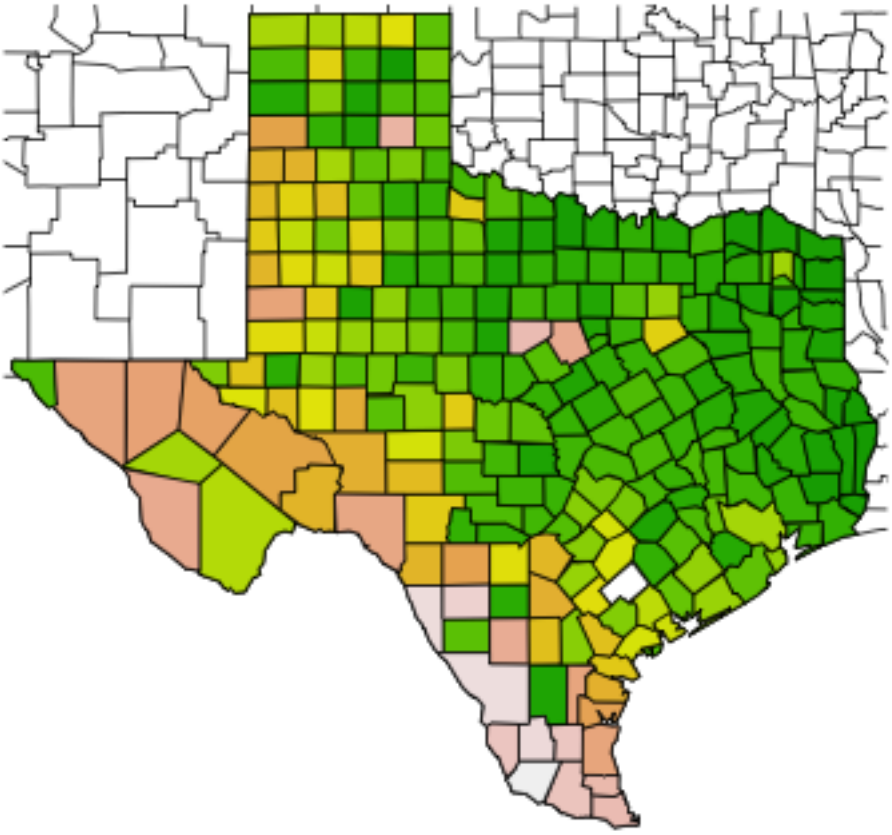
```
library(plyr)
load(url("http://www.stat.cmu.edu/~ryantibs/statcomp/exams/censusCleaned.RData"))
dim(census)
```

```
## [1] 70877    31
```

In what follows, you will fill in missing pieces of various functions that are designed to ultimately plot a county-level color map of a particular US state. The colors in the map will reflect the values of a particular numeric variable (column) available in the `census` data frame. For example, after finishing this question, a call to

```
my.state.map(census, bb.df, state="Texas", var="pct_Hispanic_ACS_09_13")
```

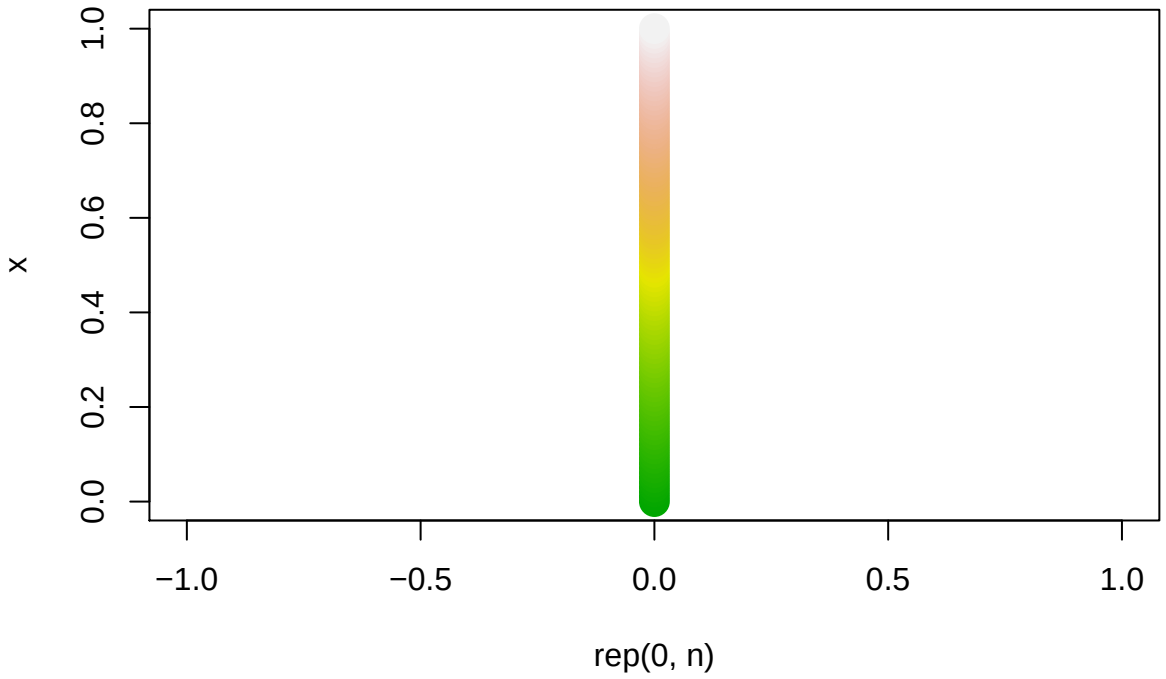
should produce the following plot:



Setup: color wheel. Let's begin by simply demo-ing the function `my.color.fun()`, which you will use to color in the various counties in your eventual plot. This function takes as input a vector `x` of values between 0 and 1, and returns a vector of colors that lie on a specturm between green and white. Check it out:

```
my.color.fun = function(x) {
  fun = colorRamp(terrain.colors(50))
  cols = rgb(fun(x),max=255)
  return(cols)
}

# Demo the color function
n = 100
x = seq(0,1,length=n)
x.col = my.color.fun(x)
plot(rep(0,n),x,pch=19,cex=2,col=x.col)
```



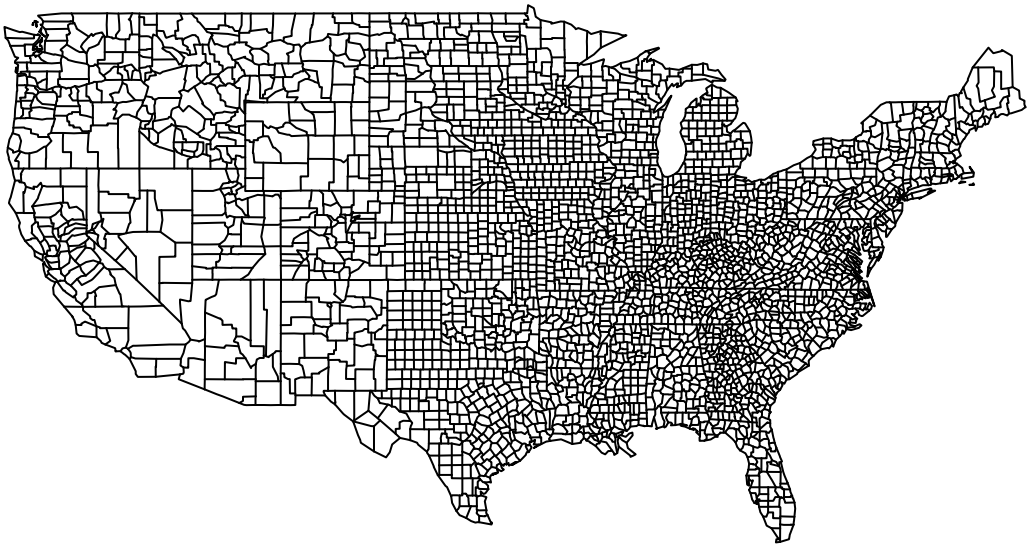
As you can see above, we have colored in points according to their height, and green corresponds to low values, while white corresponds to high values. To reiterate, `my.color.fun()` only takes values **between 0 and 1**.

Setup: maps and boundaries. Now let's load the `maps` package (this will need to be installed prior to calling `library()`), and try a few demo plots. In the first plot, we're going to plot all the counties in the US; in the second, we're going to limit the viewable longitudes and latitudes by setting the `xlim` and `ylim` arguments:

```
library(maps)
```

```
##
## Attaching package: 'maps'
##
## The following object is masked from 'package:plyr':
##
##     ozone
```

```
map("county")
```



```
map("county", xlim=c(-124.48200,-114.13121), ylim=c(32.52883,42.00952))
```

The first plot looks cool, but is far too busy to see individual county-level detail. We can see that the second plot has picked out a **bounding box**, in terms of longitudes and latitudes, for the state of California. Here we just happened to know that the longitudes for California were in between -124.48200 (western most part) and -114.13121 (eastern most part), and the latitudes for California were in between 32.52883 (southern most part) and 42.00952 (northern most part).

How did we know such numbers? We computed them from a data table of bounding boxes for **all counties in the US**. This data table is available on the course website, and the code below reads it in:

```
bb.df = read.csv("http://www.stat.cmu.edu/~ryantibs/statcomp/exams/bounding.csv")
dim(bb.df)
```

```
## [1] 3221   11
```

```
head(bb.df)
```

```
##   statefips countyfips    name corner_sw_lat corner_sw_lon corner_nw_lat
## 1         1          1 Autauga      32.30757     -86.92124      32.70821
## 2         1          3 Baldwin      30.14656     -88.03731      31.31902
## 3         1          5 Barbour      31.61756     -85.74843      32.14825
## 4         1          7    Bibb      32.83152     -87.42199      33.24693
## 5         1          9  Blount      33.76517     -86.96336      34.26048
## 6         1         11 Bullock      31.88028     -85.99926      32.30529
##   corner_nw_lon corner_ne_lat corner_ne_lon corner_se_lat corner_se_lon
## 1     -86.92124      32.70821     -86.41117      32.30757     -86.41117
## 2     -88.03731      31.31902     -87.36659      30.14656     -87.36659
## 3     -85.74843      32.14825     -85.04882      31.61756     -85.04882
## 4     -87.42199      33.24693     -86.87602      32.83152     -86.87602
## 5     -86.96336      34.26048     -86.30352      33.76517     -86.30352
## 6     -85.99926      32.30529     -85.41004      31.88028     -85.41004
```

Each row in the data frame `bb.df` corresponds to a particular county in the US. The first 2 columns give the state FIPS code and county FIPS code. The next 8 columns give the latitudes and longitudes for the 8 corners of the bounding box of the county: southwest, northwest, northeast, and southeast corners, respectively. You will want to use this data, on bounding boxes for counties, to get bounding box for states (each state being comprised, of course, of many counties).

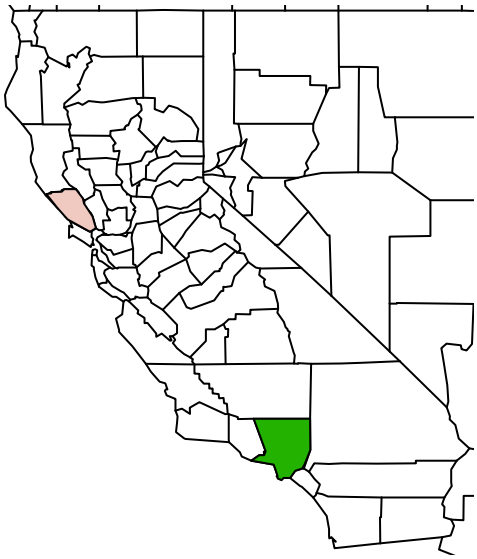
Part a: bounding boxes for states. Fill in the following function to get a bounding box for a particular state, given the `census` data frame, the `bb.df` data frame, and the state name `state` (this name is assumed to match one of the names in the `census$State_name` column).

```
my.state.lims = function(census, bb.df, state) {
  # TODO: compute the proper x and y limits
  xmin = 0; xmax = 0; ymin = 0; ymax = 0;
  return(c(xmin,xmax,ymin,ymax))
}
```

Test it out with `state="California"`. It should match the numbers given above.

Part b: region names. We have bounding boxes for states, so we can set the right plot limits with `map()`, but how about the colors? For this we use the following commands:

```
map("county", xlim=c(-124.48200,-114.13121), ylim=c(32.52883,42.00952))
map("county", add=TRUE, fill=TRUE, col=my.color.fun(c(0.1,0.9)),
    region=c("california,sonoma","california,los angeles"))
```



The first call to `map()` produces our map of California with black boundaries between counties and no colors. The second call to `map()` draws on top of this map (note the argument `add=TRUE`). We set `fill=TRUE`, and then specify two colors through the `col` argument (here, a green and a pink), and then two **region names** through the `region` argument (here Sonoma County and Los Angeles County). Note the **format required** for the region name by the `map()` function:

- all lowercase letters;
- state name first, followed by a comma;
- then county name, without a preceding space.

Other examples include "california,san francisco" (for San Francisco County in California) or "pennsylvania,allegheny" (for Allegheny County in Pennsylvania).

Fill in the following function to get the region names for all counties in a particular state, given the `census` data frame, and the state name `state` (again, this name is assumed to match one of the names in the `census$State_name` column). The returned region names must conform to the format specified above, so that they can be eventually passed to `map()`.

```
my.region.names = function(census, state) {
  # TODO: figure out the region names
  region.names = ""
  return(region.names)
}
```

Part c: county-level averages. We can now plot the counties in a particular state, and also color each one with a (possibly different) color of our choosing, by setting the `col` and `region` arguments in `map()`. Here we are interested in making those colors informative, by having them reflect the level of some variable that has been measured in the `census`; in particular, we're going to consider columns 8 through 31 as candidate variables for this task.

Let's take the 8th column, called `Med_HHD_Inc_ACS_09_13`, as an example. Looking at the values of this column across all census tracts in California:

```
dat = subset(census, subset=State_name=="California", select="Med_HHD_Inc_ACS_09_13")
summary(dat)
```

```
##  Med_HHD_Inc_ACS_09_13
##  Min.   : 10509
##  1st Qu.: 43463
##  Median : 60361
##  Mean   : 66574
##  3rd Qu.: 82996
##  Max.   :250001
```

We can see that there are 7802 census tracts in California, and for these, the median household income ranges from $2499 in one census tract all the way up to $250,001 in another. (Pause: wow! That's a huge variation...) We'd like to somehow plot county-specific median household incomes on our map of California. For this we need to do two things:

1. for each county in California, compute the **average median household income** among all its census tracts;
2. scale these values, computed in step 1, to lie between 0 and 1.

A word about step 1: we actually want to perform **population-weighted county averages** rather than pure averages. Given numbers $x_1, \ldots x_n$ and weights $w_1, \ldots w_n$, we can define a weighted average of the x_i's by:

$$(w_1 \cdot x_1 + \ldots + w_n \cdot x_n)/(w_1 + \ldots + w_n).$$

Here, we are asking for the same thing, but just asking for the weights to be defined by the populations of the various census tracts.

Fill in the function below, which performs county-level averages of a particular census variable, for all counties in a particular state. The arguments passed in are the `census` data frame, the state name `state` (this name is assumed to match one of the names in the `census$State_name` column), and the variable name `var` (this name is assumed to match one of the names of the columns 8 through 31 of `census`). The function must perform population-weighted averages, and it must then scale these average values to lie between 0 and 1.

```
my.county.avgs = function(census, state, var) {
  # TODO: compute these county-level averages, then scale to lie in [0,1]
  avgs.scaled = 0
  return(avgs.scaled)
}
```

Part d: finally, the maps! Last but not least, we want to write a function that ties in all these tasks together. Fill in the function below, which given the `census` data frame, a state name `state` (assumed to be found in the `census$State_name` column), and the variable name `var` (assumed to match one of the names of the columns 8 through 31 of `census`), plots a map of the counties in this state, with the colors corresponding to the county-level averages of the variable in question. First, the code to be completed, and then some comments afterward:

```
my.state.map = function(census, bb.df, state="Pennsylvania",
                        var="Med_House_value_ACS_09_13") {
  ## TODO: error checking here

  # From part a, xlim and ylim
  lims = my.state.lims(census,bb.df,state)

  # From part b, region names
  region.names = my.region.names(census,state)
```

```
  # From part c, county-level averages, scaled
  avgs.scaled = my.county.avgs(census,state,var)

  # TODO: plot the map!

  invisible(avgs.scaled)
}
```

You must fill in two parts of code above. First, you must perform a bunch of error checking on the inputs, namely, on the `state` and `var` arguments:

- if `state` is not one of the names found in `census$State_name`, then you must exit the function and throw an error to this effect;
- if `var` is not one of the names found in the columns 8 through 31 of `census`, then you again must exit and throw an error to this effect;
- if `state` is "Alaska", "Hawaii", or "District of Columbia", then you must exit and throw an error (since the `map()` function can't handle plotting the counties in these states).

(*Hint*: recall `stop()`, or `stopifnot()`.)

Second, you must fill in the plotting code, to produce a map of the desired state, where the counties are colored in according to the `my.color.fun()` function, as applied to the county-level scaled averages of the variable in question. (*Hint*: this will look similar to the sample code in part b, above.) Importantly, if any counties are missing (i.e., they may have been thrown out of the `census` data frame in the data cleaning process in question 1 of the exam) then they should be colored in gray.

Once completed, you can test your function out by calling `my.state.map(census, bb.df, state="Texas", var="pct_Hispanic_ACS_09_13")`, and compare to the graphic given at the beginning of this bonus problem.

Part e: choose your team name, and compete! This bonus problem will be graded using a class-wide friendly competition. Decide on a "team name"—which can be completely self-identifying or non-identifying, up to you—and enter it below. If you choose to participate in this bonus, then we will run live tests on the last day of class on your function `my.state.up()`, so make sure you code it up precisely as instructed (and of course, it goes without saying, make sure your submission knits ...)!

My team name is [SOMETHING BRILLIANT HERE]