

CHAPTER 5

Maps

Purpose: This chapter demonstrates how to create several kinds of geographical maps using PROC GMAP.

Basic Maps

This first example demonstrates how to create a simple choropleth map, and color the areas (with gradient shading) based on your numeric response data.

Let's use the SASHELP.US_DATA -- this a fairly new data set that just started shipping in SAS 9.4. If you have an older version of SAS, you can contact SAS Tech Support to find out how to download this new data. The data contains various demographic information about each of the U.S. states, such as population for year 2010 (the variable is named POPULATION_2010). To make the numbers a little more manageable, let's convert the values to millions (ie, divide them by 1,000,000).

```
data my_data; set sashelp.us_data;
format millions comma8.1;
millions=(population_2010/1000000);
run;
```

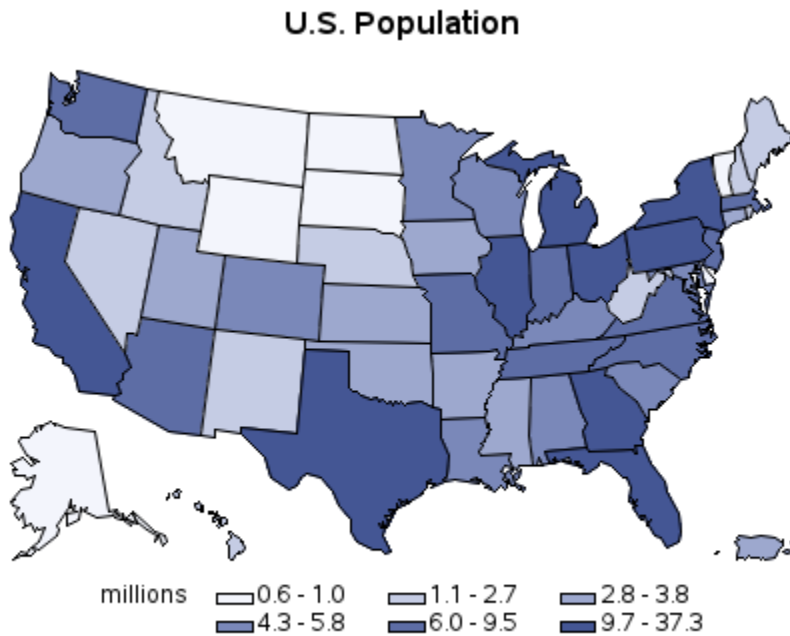
Here's what the first few rows of the response data look like:

STATENAME	STATECODE	millions
Alabama	AL	4.8
Alaska	AK	0.7
Arizona	AZ	6.4
Arkansas	AR	2.9
California	CA	37.3

The areas (such as states) in maps are identified by a variable in the data set. In many cases this variable is called ID, but in MAPS.US it is called STATECODE. Whatever the variable is named, it must be the same in both your map and response data sets (for example, we have a variable named STATECODE in both our map and response data). We can plot the population values on a U.S. map using the following very simple code.

```
title1 ls=1.5 "U.S. Population";  
proc gmap data=my data map=maps.us;  
id statecode;  
choro millions;  
run;
```

Here is the resulting map, using all default settings. Note that the default colors come from the ODS style that is in use (in this case, the HTMLBLUE style):

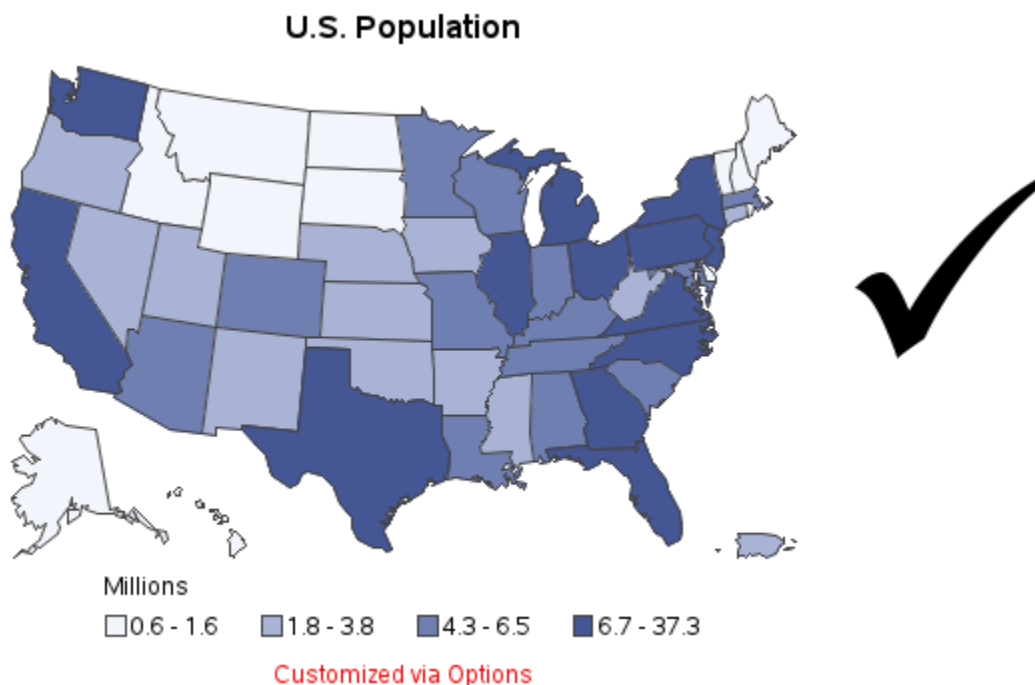


Default Map

The default code produces an OK map (above), but we can easily specify a few options to improve it. For example, it is a bit difficult to distinguish six shades of blue, so let's use the LEVELS=4 option to divide the states into four color groups (quartiles) instead of the default six groups. Also, the shape of the color chips in the legend is not aesthetically pleasing, therefore let's use a LEGEND statement to specify a more square shape, and place a custom legend label above the legend (rather to the left).

```
legend1 label=(position=top 'Millions')
shape=bar(.1in,.1in);
title1 ls=1.5 "U.S. Population";
proc gmap data=my_data map=maps.us;
id statecode;
choro millions / coutline=gray44
levels=4 legend=legend1;
run;
```

The resulting slightly customized map looks great, and was produced with only a few lines of code. The general map user can probably produce most of the maps they'll ever need, by re-using this simple code (above).



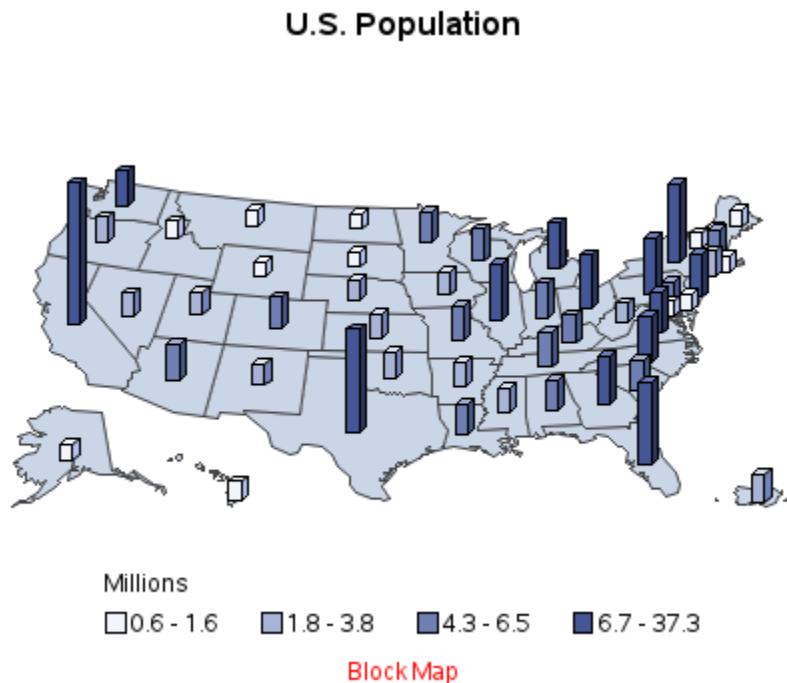
3D Maps

Now that you know the basics, you're probably eager to create some *fancy* maps. I contemplated leaving this section out (for your own good), but I finally opted to include it ... more as a 'warning' of what not to do, than an example of what to do. This example

demonstrates how to create three different kinds of 3D maps, and cautions you of the problems inherent in each of the three.

The first example is a BLOCK map, where each map area has a 3D bar with the height and color of the bar representing the response data. Of the three 3D maps, this one is the “lesser of evils,” but still bad enough that I recommend you not use it. The main problem is that tall bars towards the front of the map are likely to obscure shorter bars in the back of the map. Also, by default the heights of the bars are not scaled relative to zero (this is similar to a bar chart not starting at zero). Also, one side of the bar is a darker/shadow shade of the bar color, and that shadow color might be confused for the gradient color of another bar.

```
title1 ls=1.5 "U.S. Population";  
legend1 label=(position=top 'Millions')  
  shape=bar(.1in,.1in);  
proc gmap data=my_data map=maps.us;  
id statecode;  
block millions / coutline=gray44  
  levels=4 legend=legend1;  
run;
```



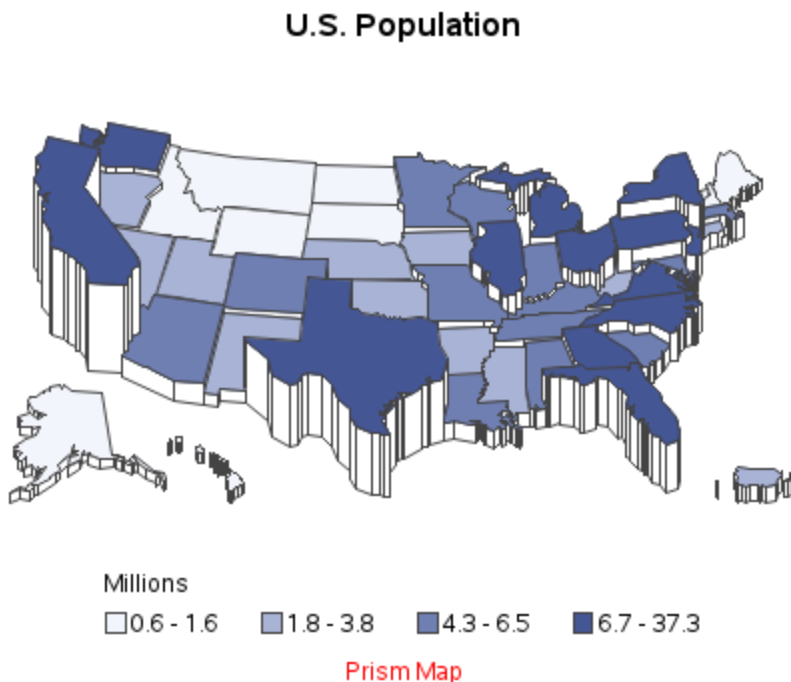
Next is the PRISM map, where each of the colored geographical areas in the map is extruded (or raised up) to represent the data values. The higher the value, the higher the area is raised. Similar to the block map, the main problem with the prism map is that when tall areas happen to be in front of short areas, the shorter areas are hidden. The side walls of the extruded areas can also be distracting (the more detail in the map area, the more lines in the extruded sides).

```

title1 ls=1.5 "U.S. Population";
legend1 label=(position=top 'Millions')
  shape=bar(.1in,.1in);
proc gmap data=my_data map=maps.us;
id statecode;
prism millions / coutline=gray44
  levels=4 legend=legend1;
run;

```

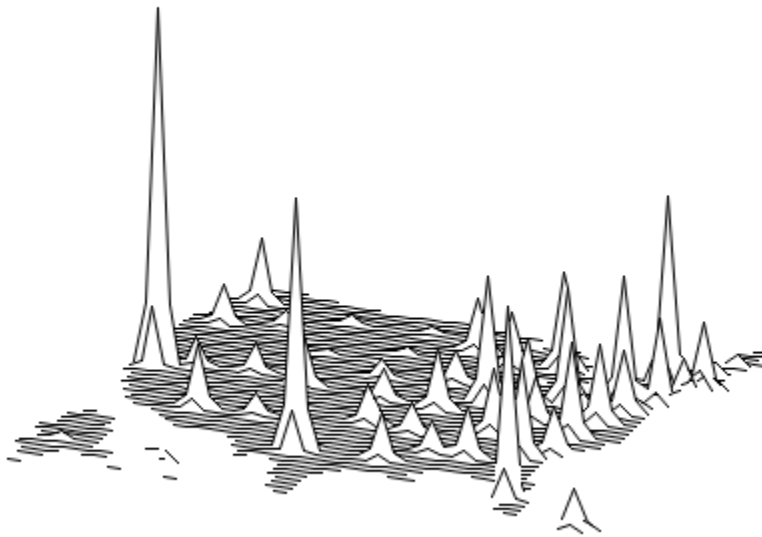
To make my point – can you identify all 50 U.S. states in the map below? Take as much time as you need! ☺



And, last but not least, the SURFACE map. It basically takes the block (bar) map, and drapes a surface over the tops of all the 3D bars. This map looks interesting, but has little analytic value. At best, it is a very vague and abstract representation of the data. There's no way to really tell which 'peak' represents which state, and it's very difficult to compare the heights of the peaks.

```
title1 ls=1.5 "U.S. Population";  
proc gmap data=my_data map=maps.us;  
id statecode;  
surface millions;  
run;
```

U.S. Population



Surface Map

Custom Color Binning

PROC GMAP's quantile binning (controlled via the LEVELS= option) usually does a pretty good job of splitting the data into the desired number of 'bins' (or 'buckets'), and then representing each bin with a color in the map and legend. But sometimes you don't really want the data to be evenly split – you might want the colors to be mapped to specific (meaningful) ranges of values. You want a way to put each map area into a specific color bin (ie, custom color binning) based on its data value.

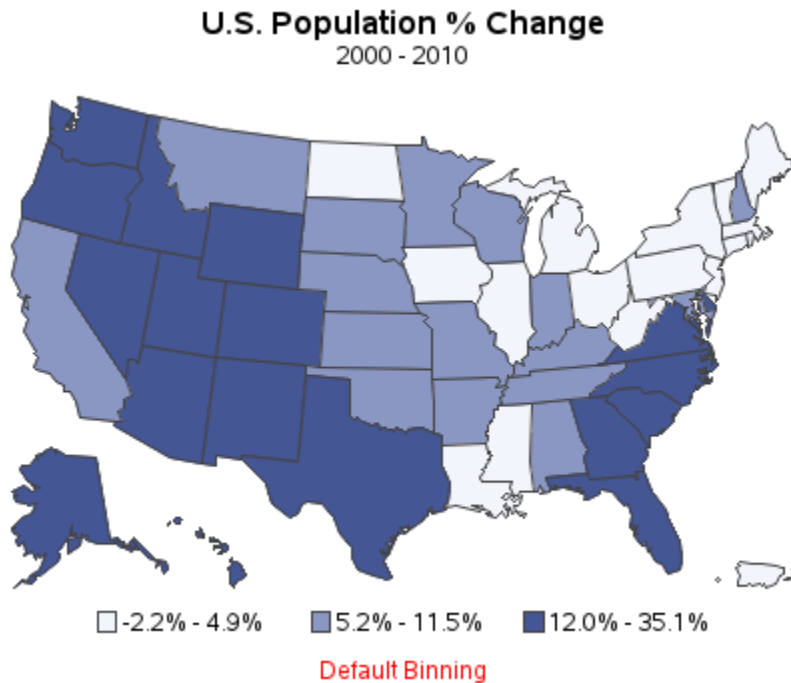
Let's start with some data that shows the % change in population between the years 2000 and 2010 for each state. First, we'll make a copy of the data and convert the numbers so they are true percent values, and assign a PERCENT format. Here's the code to perform the conversion, and what the first few rows of resulting data look like.

```
data my_data; set sashelp.us_data;
format change percentn7.1;
change=change_2010/100;
run;
```

STATENAME	STATECODE	change
Alabama	AL	7.5%
Alaska	AK	13.3%
Arizona	AZ	24.6%
Arkansas	AR	9.1%
California	CA	10.0%

Let's start with the default color binning, and then modify it to create a map with custom binning. If we use the built-in LEVELS=3 option, PROC GMAP's quantile binning creates color bins that each contain about 1/3 of the states (similar to the first example in this chapter).

```
title1 ls=1.5 "U.S. Population % Change";
title2 "2000 - 2010";
legend1 label=none shape=bar(.1in,.1in);
proc gmap data=my_data map=maps.us;
id statecode;
choro change / coutline=gray44
levels=3 legend=legend1;
run;
```



The resulting map (above) is useful, but the color bins could be a bit more meaningful. For example, the bin for the lowest 1/3 of the states covers the range -2.2% to 4.9%, and makes no distinction between negative and positive change. Perhaps it would be more interesting to see the states with a negative change (ie, population loss) plotted in a different color from the states with a positive change (ie, population growth), and so on.

One way to do this is to perform custom binning (sometimes called bucketing) in a data step, and then plot those custom bins (or buckets) on the map. Here is some code to create three custom bins: A, B, and C. I store them in a variable called BUCKET (you could use any variable name you want).

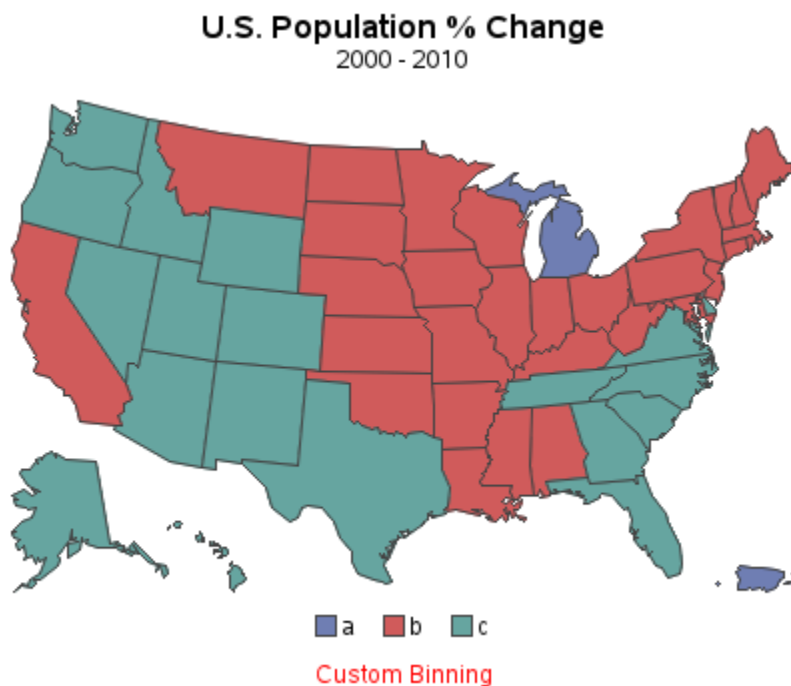
```
data my_data; set my_data;  
if change<0 then bucket='a';  
else if change<=.10 then bucket='b';  
else if change>.10 then bucket='c';  
run;
```

Now we can easily plot those bucket values on the map. Note that I specify all the possible bucket values in the MIDPOINTS option – this guarantees that they will all show up in the legend, even if the response data does not have any rows with matching values. This is important if you are going to re-use the same code with different response data, and guarantees that the same values will always be mapped to the same colors.


```

title1 ls=1.5 "U.S. Population % Change";
title2 "2000 - 2010";
legend1 label=none shape=bar(.1in,.1in);
proc gmap data=my_data map=maps.us;
id statecode;
choro bucket / coutline=gray44
midpoints = 'a' 'b' 'c' legend=legend1;
run;

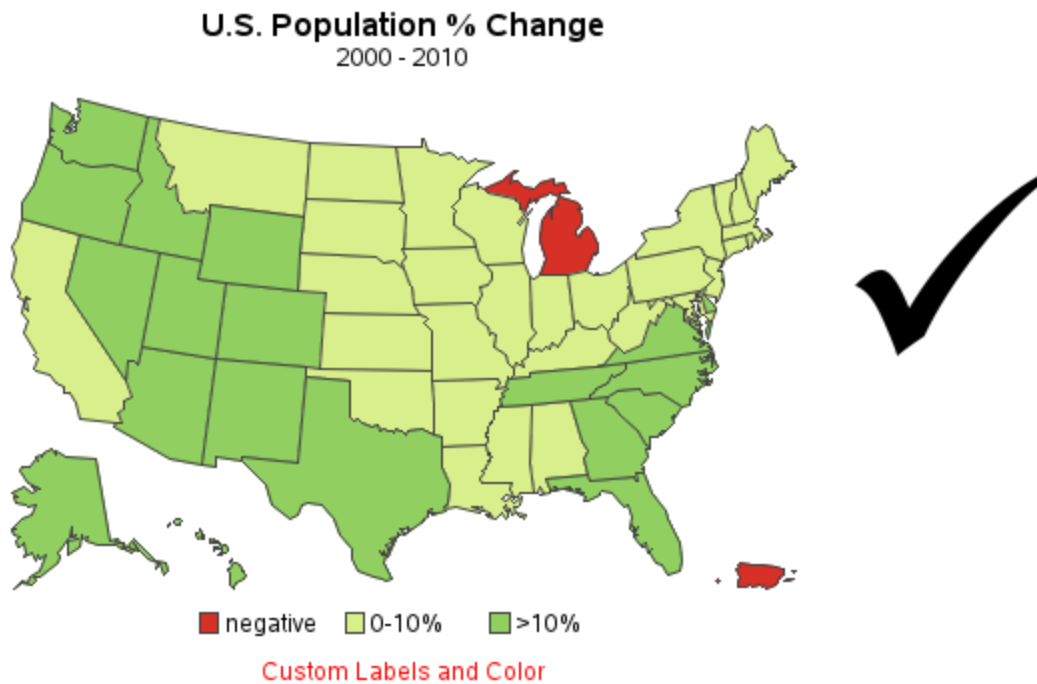
```



The bucket names in the legend are not all that useful, but we can replace that with some more meaningful text. To keep this example simple, we will hard-code the text in the legend statement. Slightly more complex code could create user-defined formats, which would be more data-driven and more easily generalizable (see the book *SAS/GRAPH: Beyond the Basics*). We'll also specify some custom colors in PATTERN statements, instead of using the default colors from the ODS style.

```
title1 ls=1.5 "U.S. Population % Change";
title2 "2000 - 2010";
pattern1 v=s color=cxD73027; /* red */
pattern2 v=s color=cxD9EF8B; /* light green */
pattern3 v=s color=cx91CF60; /* darker green */
legend1 label=none shape=bar(.1in,.1in)
      value=(justify=left t=1 "negative" t=2 "0-10%" t=3 ">10%");
proc gmap data=my_data map=maps.us;
id statecode;
choro bucket / coutline=gray44
midpoints = 'a' 'b' 'c' legend=legend1;
run;
```

And now we have a nice map, with custom binning, colors, and labels that are meaningful, and chosen specifically for this data.



Blank Maps

The previous examples demonstrated how to plot response data on a map, whereas this example demonstrates how to draw just the outline of the areas in a map with no data. This can be very useful when you want to see what a map looks like, or sometimes you might have a use for just an empty outline map.

When you're not plotting any response data on the map, you can use the map data set for both the MAP= and DATA=. We'll use MAPS.US for the first few blank map examples, since you're already familiar with it. You can specify any numeric variable that happens to be in the data set for your response variable – most maps have a numeric SEGMENT variable, so let's use that one. And since we want the entire map to be filled with the same color, specify LEVELS=1.

```
title1 ls=1.5 "MAPS.US";  
pattern1 v=s color=white;  
proc gmap data=maps.us map=maps.us;  
id statecode;  
choro segment / outline=gray44  
levels=1 nolegend;  
run;
```

Here is the resulting map – very simple, very easy.



Blank Map

Subsetting Maps

Sometimes the maps that SAS ships are not exactly what you want. For example, maybe you don't want Puerto Rico in your U.S. map. Fortunately the SAS maps are stored in data sets, and you can manipulate them just like any other data set – either by applying a WHERE clause in a data step to create a new map, or by applying a WHERE clause to the map when PROC GMAP is run (I usually prefer the latter).

I'm creating blank maps here to keep the code short, but you can use the same technique when you're plotting data/colors on the map.

In this first example, we'll create a map of the contiguous U.S. states, by eliminating Puerto Rico (PR), Alaska (AK), and Hawaii (HI) from the map. Be careful to match up your parentheses when you add the where clause to the MAP=.

```
title1 ls=1.5 "Contiguous U.S. States";  
pattern1 v=s color=white;  
proc gmap data=maps.us  
  map=maps.us (where=(statecode not in ('PR' 'AK' 'HI')));  
id statecode;  
choro segment / coutline=gray44  
  levels=1 nolegend;  
run;
```

Contiguous U.S. States



Subset by Exclusion

When you're dealing with a small number of states, sometimes it's easier to **include** the desired areas, rather than **exclude** them. In this next example we will list the four desired states to include in the map, rather than listing the 57 areas to exclude.

```
title1 ls=1.5 "Hurricane Area";  
pattern1 v=s color=white;  
proc gmap data=maps.us map=maps.us  
  (where=(statecode in ('NC' 'SC' 'GA' 'FL')));  
id statecode;  
choro segment / coutline=gray44  
  levels=1 nolegend;  
run;
```

Hurricane Area

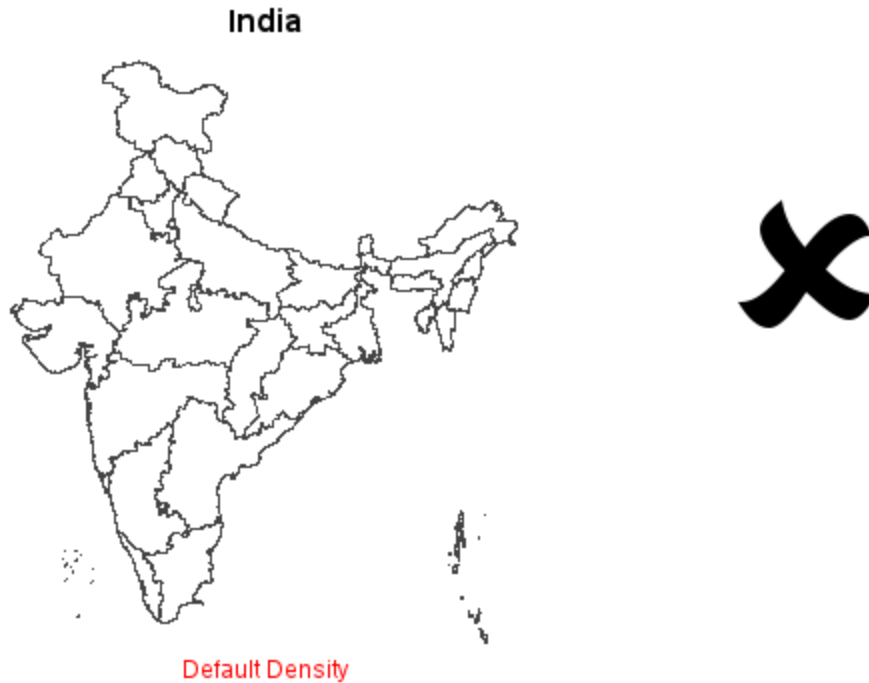


Subset by Inclusion

Simplifying Map Borders

Sometimes the borders of a map contain so many x/y points that the borders end up looking thick and/or squiggly. This problem is most prevalent for the map borders along coastlines and rivers. For example, see the borders in MAPS.INDIA, below.

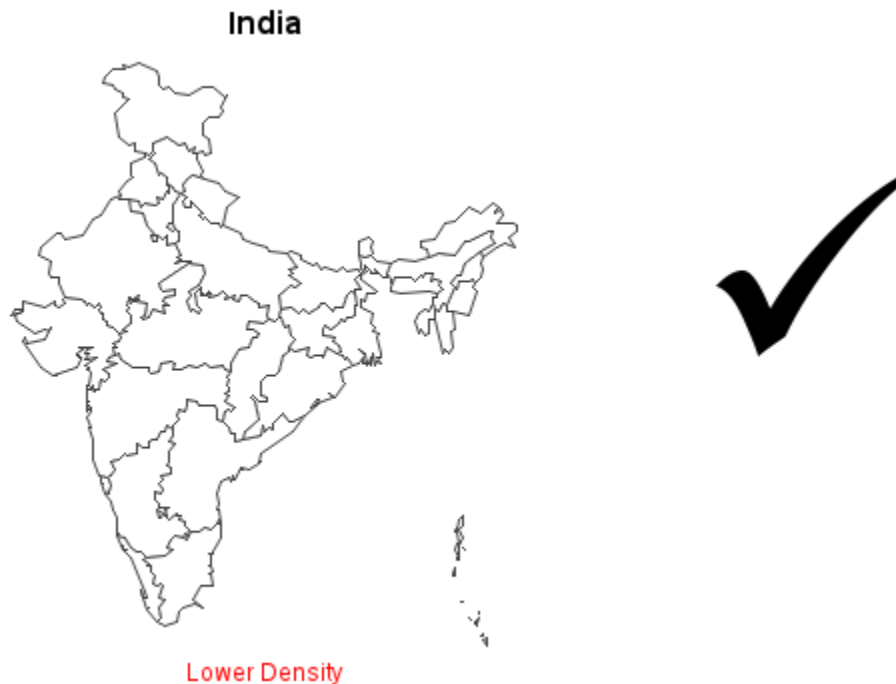
```
title1 ls=1.5 "India";  
pattern1 v=s color=white;  
proc gmap data=maps.india map=maps.india;  
id id;  
choro segment / coutline=gray44  
  levels=1 nolegend;  
run;
```



There is a DENSITY variable in most SAS maps that can be used to reduce the complexity of the borders. You can either apply a WHERE clause to the map - for example (WHERE=(DENSITY<=1)), or you can use the DENSITY option. Specify a lower density number to produce a map with simpler borders. I typically use a density value of 1 or 2, depending on what map I'm using.

```
title1 ls=1.5 "India";  
pattern1 v=s color=white;  
proc gmap data=maps.india map=maps.india density=1;  
id id;  
choro segment / coutline=gray44  
  levels=1 nolegend;  
run;
```

Notice how the map looks much less cluttered, and yet the lower density borders still do an adequate job of describing the major geographical areas in the map.








Importing Maps

What if SAS does not provide the map you need? Luckily “there’s an app for that!” ... Or more precisely, a proc. PROC MAPIMPORT allows you to import Esri shapefiles into SAS maps.

For example, let’s say you own a business in Wyoming, and you would like to see how your customers are distributed across the state. You have a customer database that contains the customer zip codes, but SAS does not supply maps of the zip code areas. The U.S. Census provides zip code maps in Esri shapefile format, which you can download, and import it into SAS.

Currently (2013), the Census URL to download zip code shapefiles is: <http://www.census.gov/cgi-bin/geo/shapefiles2010/layers.cgi>. This URL could change in the future, but you should be able to locate the new page with a little Web searching. I downloaded the “5-Digit ZIP Code Tabulation Area (2010)” for the state of Wyoming, and the file was named tl_2010_56_zcta510.zip (note that in this case ‘.zip’ means archive file format, not zip code). I created a folder called ‘Wyoming’ and extracted the zip file into that folder. Here are the files extracted:

Name	Date modified	Type	Size
 tl_2010_56_zcta510.dbf	5/10/2011 12:31 PM	DBF File	14 KB
 tl_2010_56_zcta510.prj	5/10/2011 12:31 PM	PRJ File	1 KB
 tl_2010_56_zcta510.shp	5/10/2011 12:31 PM	SHP File	9,402 KB
 tl_2010_56_zcta510.shp.xml	5/10/2011 12:31 PM	XML Document	20 KB
 tl_2010_56_zcta510.shx	5/10/2011 12:31 PM	SHX File	2 KB

We will need to run PROC MAPIMPORT twice. Once to see what the Census named their variables, and then a second time specifying the ID variable so SAS can create the map. Here's the code for the first iteration.

```
proc mapimport datafile="./Wyoming/tl_2010_56_zcta510.shp"
  out=my_map;
run;
```

There are many variables in the data set – the screen capture below shows a few of the variables. In this case, intuition tells us we are looking for a variable that contains the 5-digit zip codes, probably named something like 'ZIP' or 'ZCTA.' It appears that the Census has named their variable ZCTA5CE10.

MTFCC10	PARTFLG10	STATEFP10	ZCTA5CE10
G6350	N	56	82052
G6350	N	56	82052
G6350	N	56	82052
G6350	N	56	82052
G6350	N	56	82052

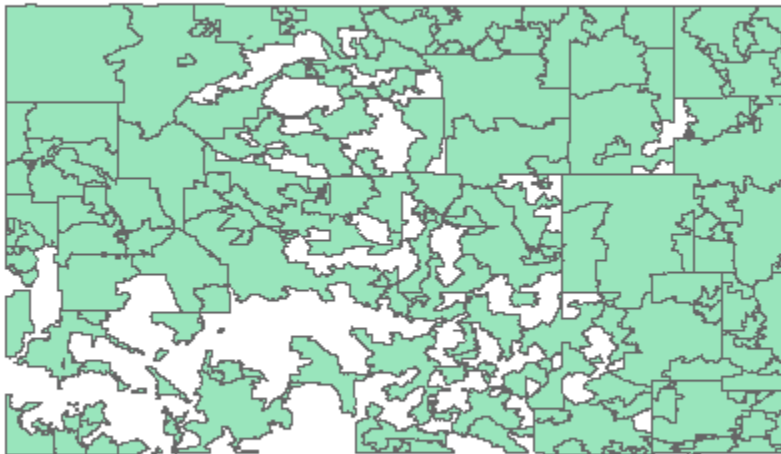
We'll now run the PROC MAPIMPORT again, this time specifying ZCTA5CE10 as the ID variable. Now that SAS knows which variable is the ID, it can group the areas and segments of the areas correctly.

```
proc mapimport datafile="./Wyoming/tl_2010_56_zcta510.shp"
  out=my_map;
  id ZCTA5CE10;
run;
```


Below is the map – almost done, but not quite. I am plotting the blank map, with no response data, to see what the areas look like. I color the map so that I can easily distinguish the areas of the state that do, and do not, have a zip code associated with them. Very remote or sparsely populated areas might not have a zip code, and therefore are not part of a zip code tabulation area, for example.

```
title1 ls=1.5 "Wyoming Zip Codes";  
pattern1 v=s color=vlig;  
proc gmap data=my_map map=my_map;  
id ZCTA5CE10;  
choro segment / coutline=gray66  
levels=1 nolegend;  
run;
```

Wyoming Zip Codes



Unprojected ZCTA Map

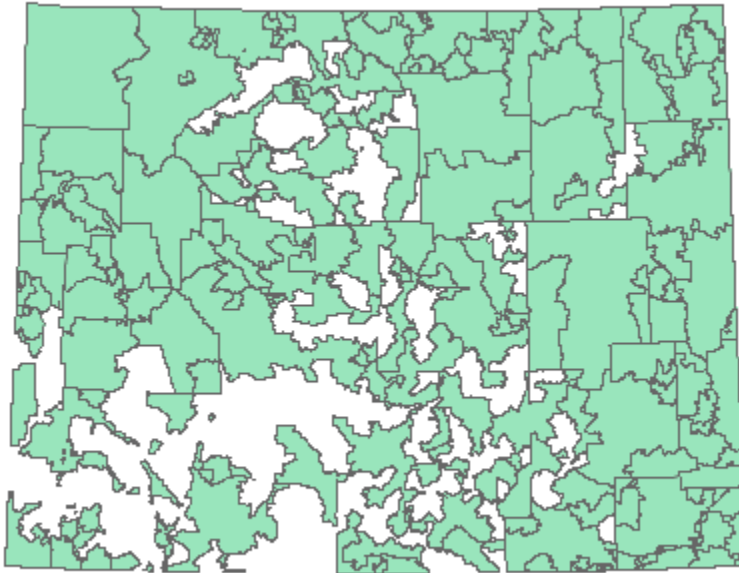
Many shapefiles, such as this one, have X and Y variables that are unprojected longitude and latitude values. When that is the case, you will need to project the coordinates in order to have a *proper* map. In general, the larger the area, the more important it is to project the map. If your map looks OK to you without projecting, and if you do not care whether you have a perfectly proper map, this step is somewhat optional. But I recommend taking the extra step, and projecting the map in most cases.

Here is the code to project the latitude/longitude coordinates in the map:

```
proc gproject data=my_map out=my_map dupok eastlong degrees;  
id ZCTA5CE10;  
run;
```

And here is the resulting map, ready for you to add your response data, and color the zip code areas by the number of customers, or any other variable you want.

Wyoming Zip Codes



Projected ZCTAMap