Most excellent figures in R with ggplot2

Ben Williams
bcwilliams@alaska.gov

Contents

1 Short version

Set up the R work environment to produce publication quality documents using ggplot. Use theme_set and ggsave to increase dpi from base R level (72 dpi), also allows for defining figure dimensions.

Load packages

```
library(extrafont)
#font_import() only do this one time - it takes a while
loadfonts(device="win")
windowsFonts(Times=windowsFont("TT Times New Roman"))

library(ggplot2)
theme_set(theme_bw(base_size=12,base_family='Times New Roman')+
  theme(panel.grid.major = element_blank(),
  panel.grid.minor = element_blank()))
```

```r
ggplot(data, aes(x=x, y=y))+geom_point()
ggsave("filename.png", dpi=300, height=4, width=4, units="in")
```

2 Basics

2.1 ggplot base figure

You say you want to create a publication quality figure using ggplot? Well then, you are in the right place! First, let's look at what ggplot produces as a base graphic. We will use the R built in data set *mtcars*.

```r
library(ggplot2)
ggplot(mtcars, aes(wt, mpg))+geom_point()
```

We get a gray background with small black dots, white grid lines, and a font that is not particularly legible when the figure is placed in this document (Figure 1a). Most journals require a figure to be 300 dpi or greater not to mention the background should be white, without grid lines, and with a font that is consistent with the rest of the manuscript. Lets address these step by step.

2.2 White background

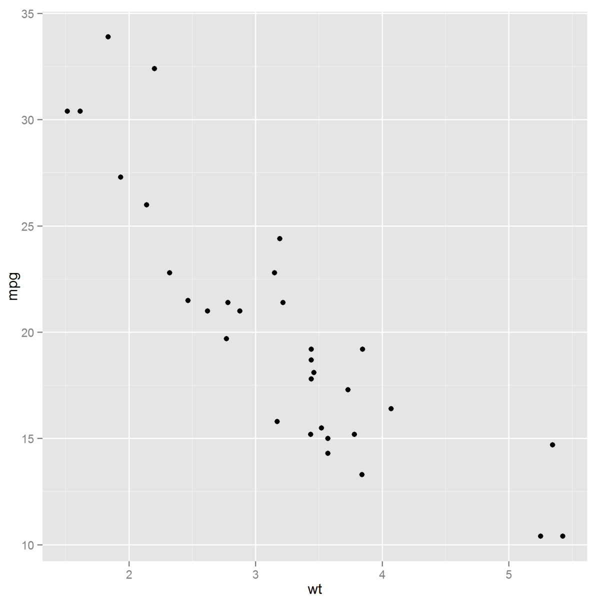
Want a white background? We've got that. Simply add a *theme* to the figure (Figure 1b).

```r
ggplot(mtcars, aes(wt, mpg))+geom_point()+theme_bw()
```

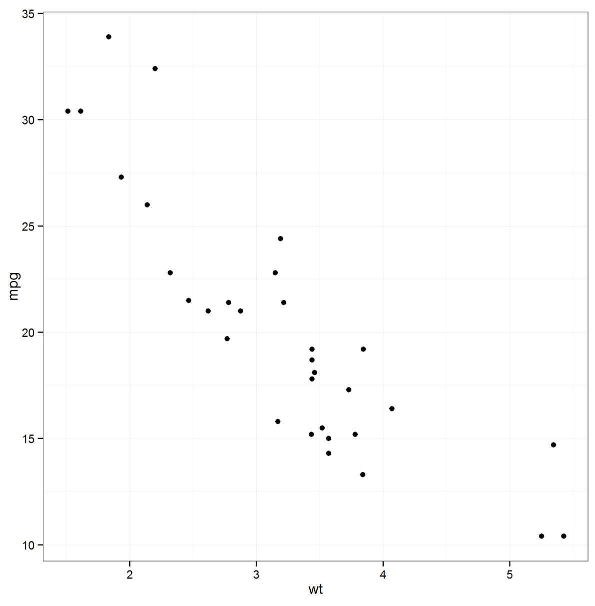
2.3 Remove grid lines

To remove grid lines add another *theme* (Figure 1c).

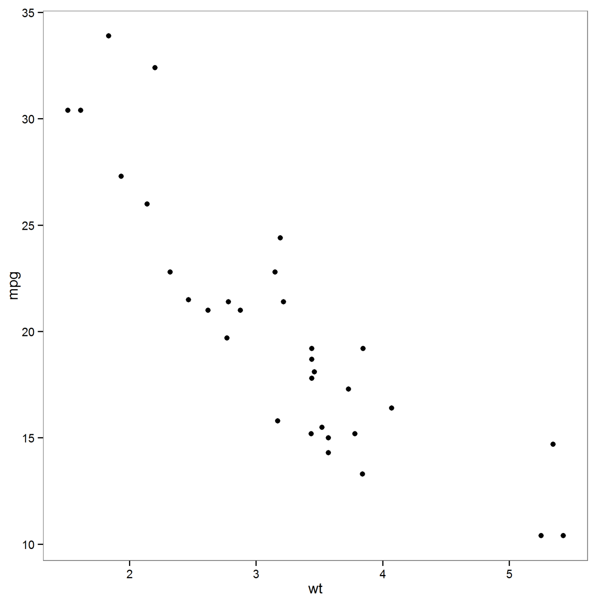
```r
ggplot(mtcars, aes(wt, mpg))+geom_point()+theme_bw()+
  theme(panel.grid.major = element_blank(),panel.grid.minor = element_blank())
```



(a) Base graphic



(b) White background



(c) White background, no grid lines

Figure 1: Remove the base color and grid lines from ggplot.

2.4 Fonts and resolution

Fonts are both easy and not so easy to deal with depending on what your needs are. In this case we want a Times New Roman 12 pt font. The 12 pt is easy so let's start with that.

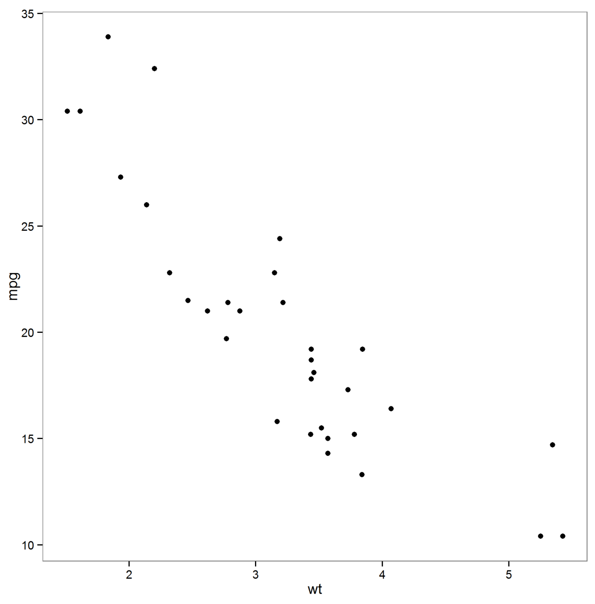
Simply add a base_size option in theme_bw() as seen in (Figure 2a).

```
ggplot(mtcars, aes(wt, mpg)) + geom_point() +
  theme_bw(base_size=12) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())
```

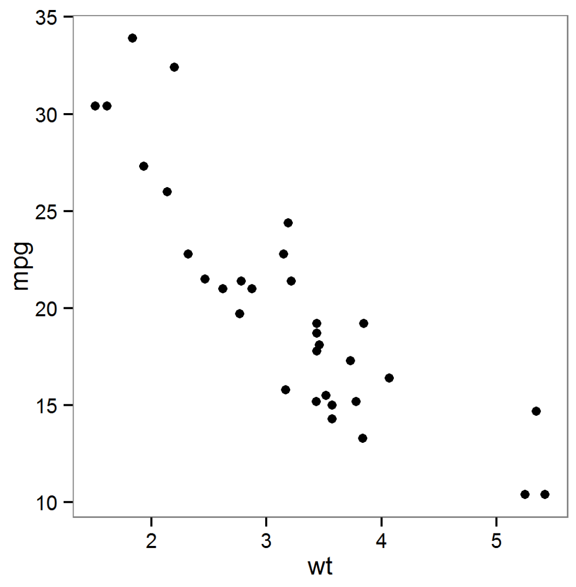
Hey nothing happened, this is the same as Figure 1c?

The reason for this is the file size. These figures were saved as .png and R has a base figure output of 72 ppi. This resolution is too low for journal submissions so let's up it. In this case I'm outputting a .png file with a set width and height and a resolution of 300. In order to save the figure we will use *ggsave* (Figure 2b).

```
ggsave("figure2b.png", dpi=300, height=4, width=4, units="in")
```



(a) Change font?



(b) 12pt font size

Figure 2: Change font and resolution.

Our figure is looking ok, but the font is not correct. I'm on a Windows machine, so these procedures may be different for other operating systems. R is not terribly great at fonts so it is necessary to define the fonts clearly. This involves loading the *extrafont* package, then importing and unpacking the fonts. *Note: Only font_import one time, it takes a while and once done you are good to go, I have it commented out as I've already run it.*

```
library(extrafont)
#font_import()
loadfonts(device="win")
```

Load the fonts for a windows device and define the font for windows.

```
windowsFonts(Times=windowsFont("TT Times New Roman"))
```

Now the font can be incorporated into the figure via the theme_bw() component (Figure 3).

```
ggplot(mtcars, aes(wt, mpg))+geom_point()+
  theme_bw(base_size=12,base_family='Times New Roman')+
  theme(panel.grid.major = element_blank(),panel.grid.minor = element_blank())
```

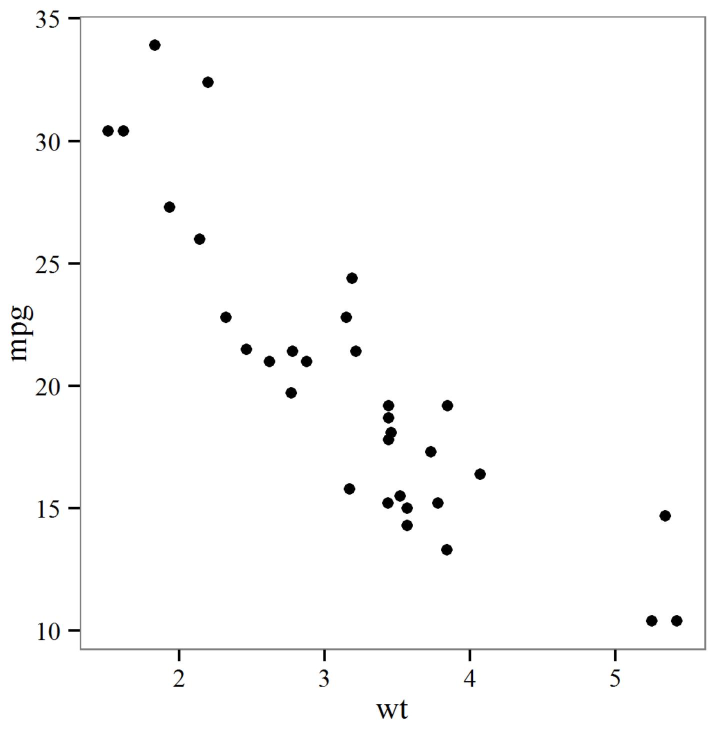


Figure 3: 12 pt Time New Roman font figure.

2.5 A better way: theme_set()

While all these adjustments to ggplot are great, there is a much better way for defining all of your ggfigures. It goes by the name *theme_set*. Here is a working example.

```
library(extrafont);library(ggplot2)
#font_import() only do this one time - it takes a while
loadfonts(device="win")
windowsFonts(Times=windowsFont("TT Times New Roman"))
theme_set(theme_bw(base_size=12,base_family='Times New Roman')+
theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank()))
```

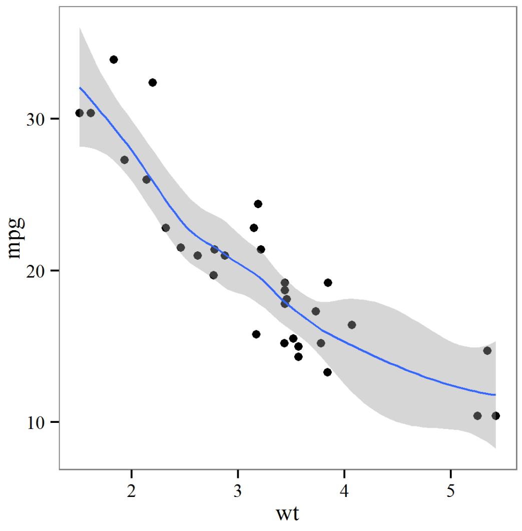
With the basic figure options set, less time is spent coding figures and more time can be spent exploring data. The code used to produce Figure 3 is now reduced to:

```
ggplot(mtcars, aes(wt, mpg))+geom_point()
```

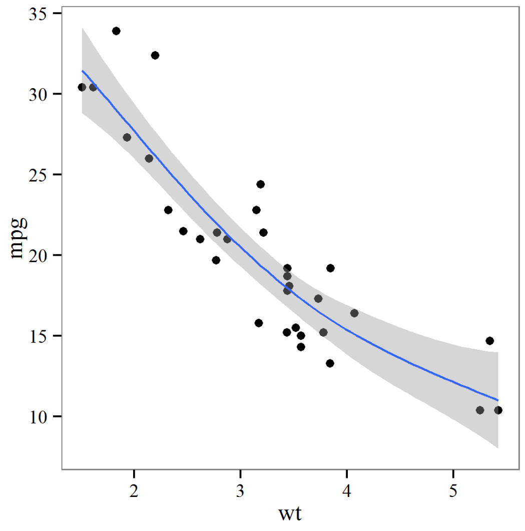
3 Smoothers

Lets explore these data, starting with a smooth. the base smooth for ggplot is “loess” (Figure 4a) for data groups <1000 and a “gam” for groups >1000 (Figure 4b).

```
options(scipen=3)
ggplot(mtcars, aes(wt, mpg))+geom_point()+geom_smooth()
```



(a) Adding a loess smoother.



(b) GAM smoother.

Figure 4: Adding smoothers (modeling).

That is great and all but most of us do not use loess on a regular basis. However we can also implement other model structures (Figure 5). In this case we added a linear regression (*lm*) and generalized linear regression with a polynomial (*glm*), a generalized additive model with knots (k) set at 5, and a non-linear least squares model. *Note that the polynomial in the glm smooth could also be done via the lm smooth (also with the poly function instead of the identity component), however I'm mostly showing that their is a glm function that can be assigned different distributions (e.g., family="binomial"), same goes for the gam smoother. More info can be found here `http://www.ats.ucla.edu/stat/r/faq/smooths.htm`. Other good things can be found on that site as well.*

```r
ggplot(mtcars, aes(wt, mpg))+geom_point()+
  geom_smooth(alpha=.1)+geom_smooth(method='lm',alpha=.1, color=2, fill=2,)+
  geom_smooth(method='glm', alpha=.1, color=3, fill=3, formula=y~x+I(x^2))+
  geom_smooth(method='gam', alpha=.1, color=4, fill=4, formula=y~s(x, k=5))+
  geom_smooth(method = "nls", formula = 'y ~ a*x^2 + b*x +c',
    start=list(a=.1,b=.5,c=.2),se = FALSE, linetype = 1,
               colour = 5,fill=5, alpha=.1)
```

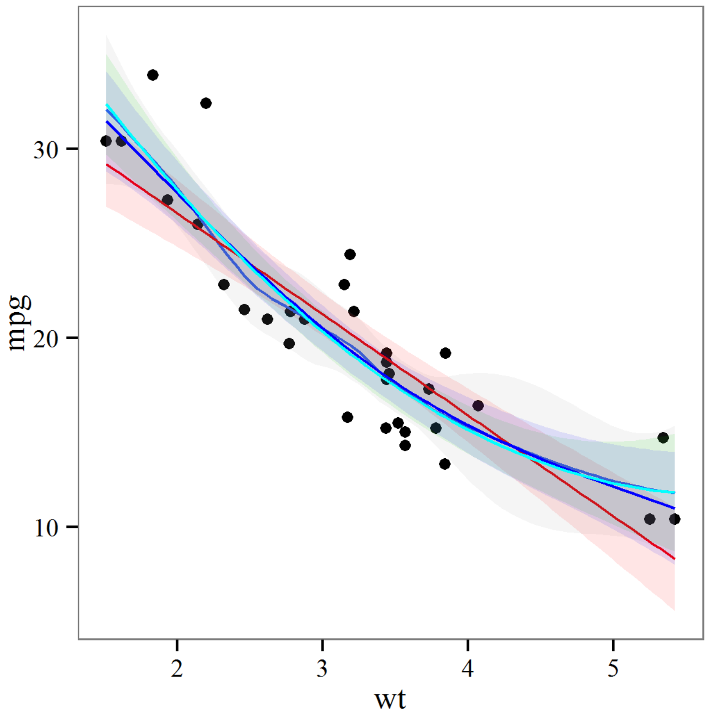


Figure 5: Multiple smoothers.

4 Plot options

4.1 Point size & color

Want bigger points, no problem simply change the size in geom_point() (Figure 6). Jitter using geom_jitter(), etc.

```r
ggplot(mtcars, aes(wt, mpg))+geom_point(size=4)+geom_smooth(alpha=.1)
```

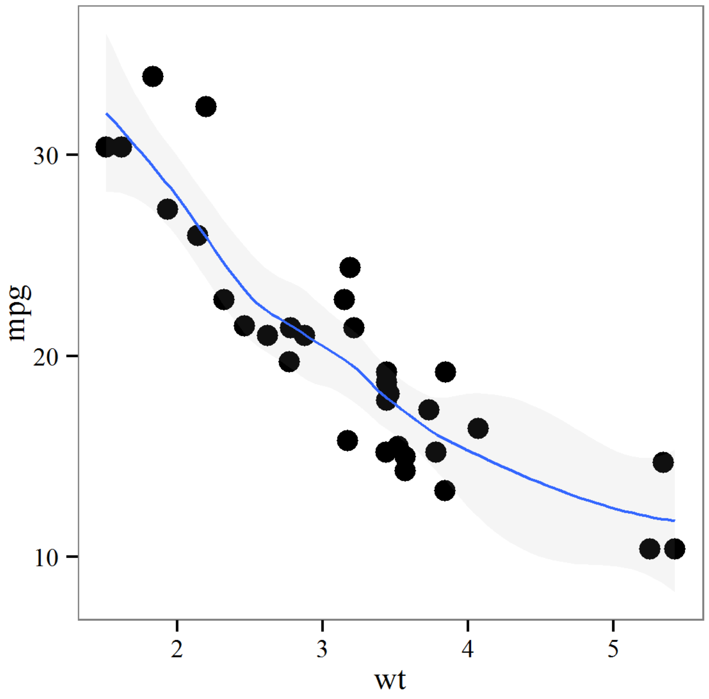


Figure 6: Point size.

However, if you want the points to be different colors this can be added to the aesthetic component (in either the ggplot() or geom_point() component). For example lets change the size of the points by a car's horsepower (hp) a continuous variable in the dataframe (Figure 7.

```
ggplot(mtcars, aes(wt, mpg))+geom_point(aes(size=hp, color=hp))+
  geom_smooth(alpha=.1)
```

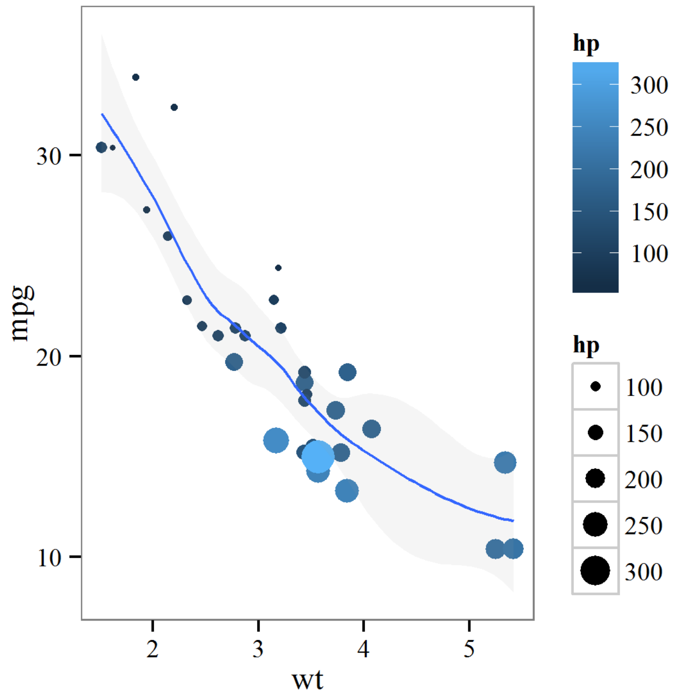


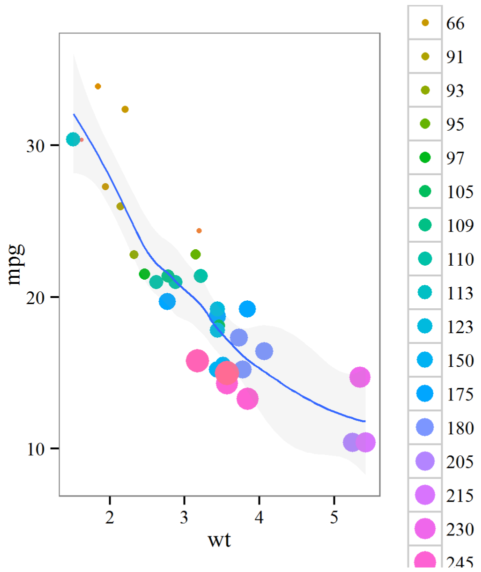
Figure 7: Multiple smoothers.

Now the colors and sizes could be determined by changing hp to a factor (Figure 8a).

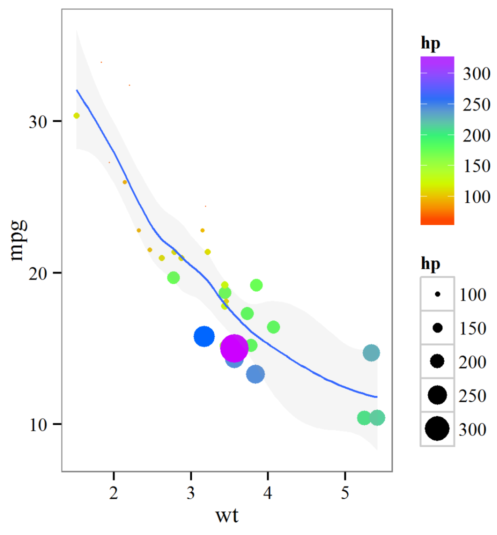
```
ggplot(mtcars, aes(wt, mpg))+geom_point(aes(size=factor(hp), color=factor(hp)))+
  geom_smooth(alpha=.1)
```

However this will often create too many discrete groups to clearly understand what is being plotted, therefore a "better" method is to define the scales for the data points. This can be done with scale_color_gradientn() (Figure 8b).

```
ggplot(mtcars, aes(wt, mpg))+geom_point(aes(size=hp, color=hp))+
  geom_smooth(alpha=.1)+
  scale_colour_gradientn(colours=rainbow(5))+
  scale_size(range=c(0,7))
```



(a) Multiple smoothers.



(b) Multiple smoothers.

Figure 8: Adding smoothers (modeling).

Want to define where the legend breaks? no problem, just add a vector of values (Figure 9).

```
bs <- seq(0,350,25)
ggplot(mtcars, aes(wt, mpg))+geom_point(aes(size=hp, color=hp))+
  geom_smooth(alpha=.1)+
  scale_colour_gradientn(colours=rainbow(5), breaks=as.vector(bs))+
  scale_size(range=c(0,7),breaks=as.vector(bs))
```

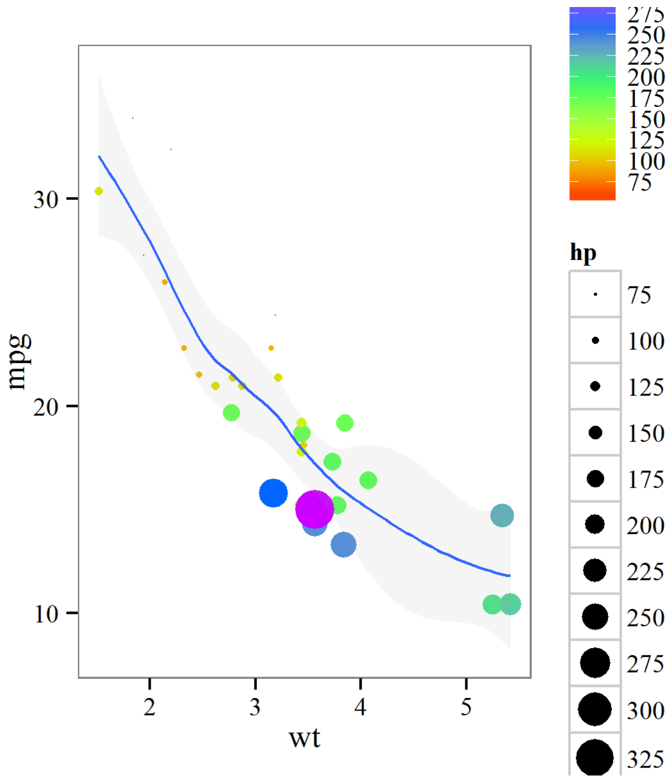


Figure 9: Multiple smoothers.

Note that the legend now falls off the plotted area for the figure. One way to fix this is to remove one of the legends, in this case the legend that references the size of the points (Figure 10).

```
ggplot(mtcars, aes(wt, mpg))+geom_point(aes(size=hp, color=hp))+
  geom_smooth(alpha=.1)+
  scale_colour_gradientn(colours=rainbow(5), breaks=as.vector(bs))+
  scale_size(range=c(0,7), breaks=as.vector(bs),guide = FALSE)
```

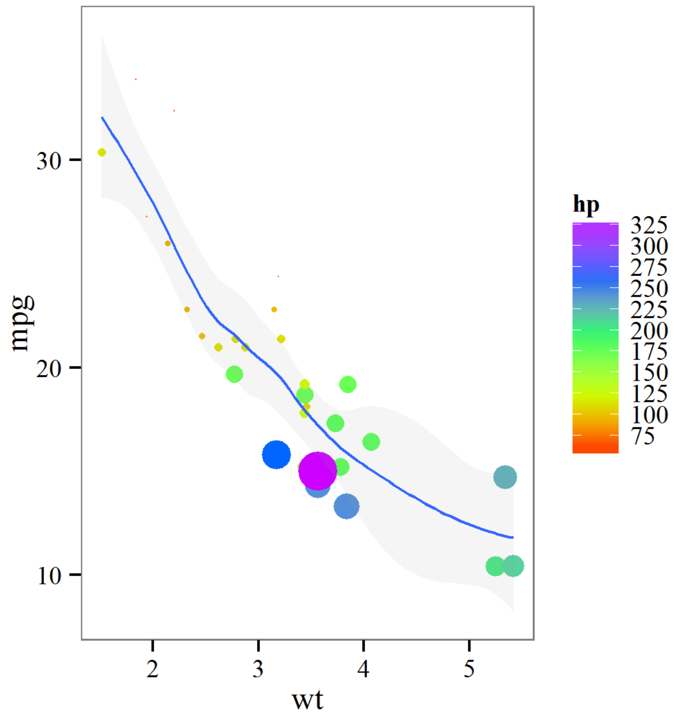


Figure 10: Combine legends.

The legend now breaks in increments of 25 using a 5 color scale from the rainbow.

4.2 Axis tickmarks

It is quite easy to change the axis tick marks. Change the y-axis to every 5th value (Figure 11).

```
ggplot(mtcars, aes(wt, mpg))+geom_point(aes(size=hp, color=hp))+
  geom_smooth(alpha=.1)+
  scale_colour_gradientn(colours=rainbow(5), breaks=as.vector(bs))+
  scale_size(range=c(0,7), breaks=as.vector(bs),guide = FALSE)+
  scale_y_continuous(breaks= seq(0,40,5))
```

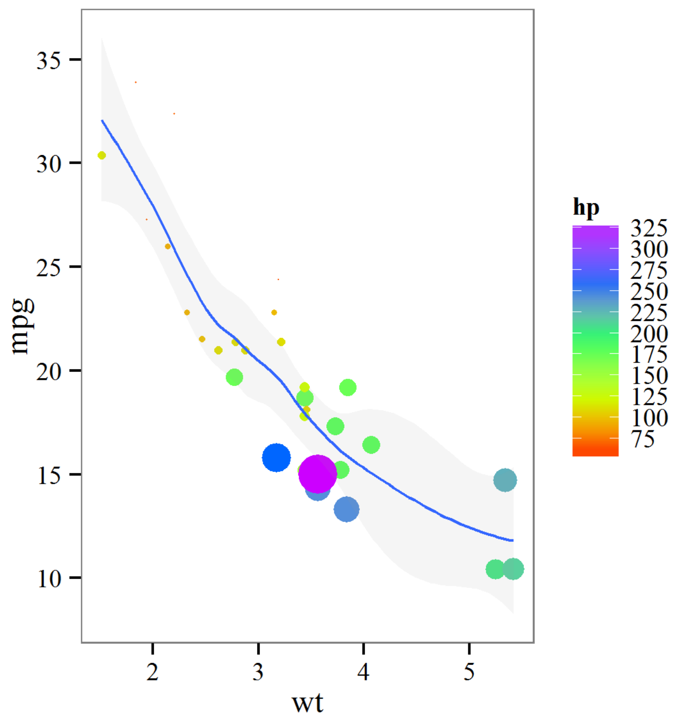


Figure 11: Change y-axis tick marks.

Also change the x-axis, this time to unequal spacings (Figure 12).

```
ggplot(mtcars, aes(wt, mpg))+geom_point(aes(size=hp, color=hp))+
  geom_smooth(alpha=.1)+
  scale_colour_gradientn(colours=rainbow(5), breaks=as.vector(bs))+
  scale_size(range=c(0,7), breaks=as.vector(bs),guide = FALSE)+
  scale_y_continuous(breaks= seq(0,40,5))+
  scale_x_continuous(breaks= c(0,1,1.5,2,3,5))
```

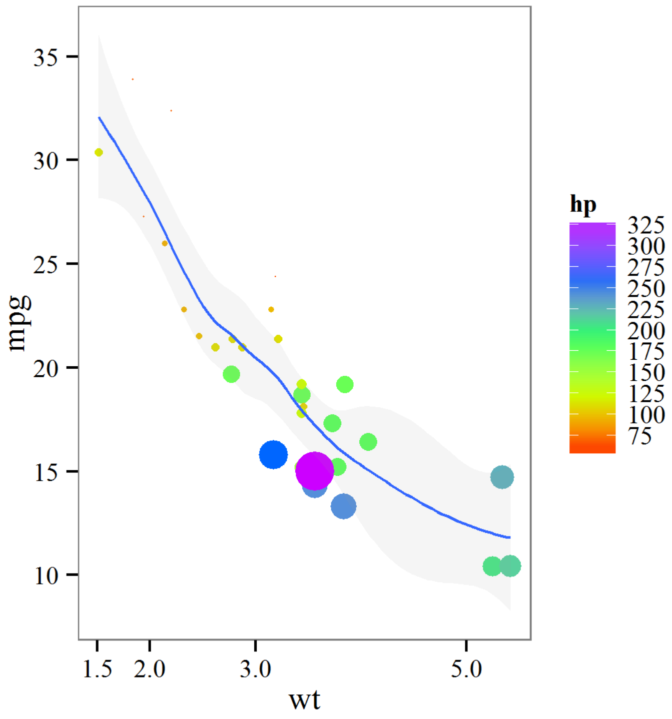


Figure 12: Change y-axis and x-axis tick marks.

Your collaborator really prefers to have all of the tick marks present so... Change the labels while keeping the tickmarks (Figure 13).

```
ggplot(mtcars, aes(wt, mpg))+geom_point(aes(size=hp, color=hp))+
  geom_smooth(alpha=.1)+
  scale_colour_gradientn(colours=rainbow(5), breaks=as.vector(bs))+
  scale_size(range=c(0,7), breaks=as.vector(bs),guide = FALSE)+
  scale_y_continuous(breaks= seq(0,40,5), labels=c(0,"",10,"",20,"",30,"",40))+
  scale_x_continuous(breaks= seq(0,5,.5), labels=c(seq(0,2,.5),"","",3.5, "",4.5,""))
```

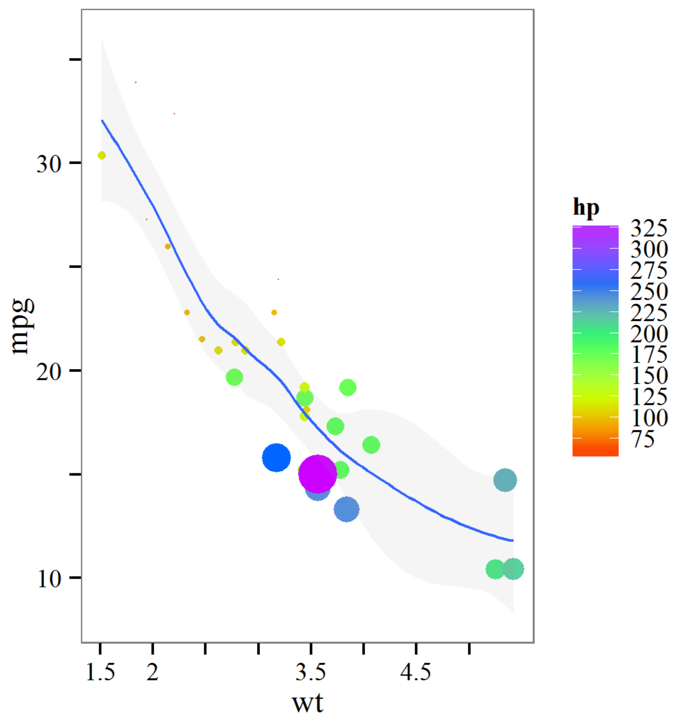


Figure 13: Change y-axis and x-axis tick marks.

However, the scale presented doesn't include zero, something that is often nice to have. The standard way to do this is to use xlim()and/or ylim().

```
ggplot(mtcars, aes(wt, mpg))+geom_point(aes(size=hp, color=hp))+
  geom_smooth(alpha=.1)+
  scale_colour_gradientn(colours=rainbow(5), breaks=as.vector(bs))+
  scale_size(range=c(0,7), breaks=as.vector(bs),guide = FALSE)+
  xlim(0,5)+ylim(0,40)
```

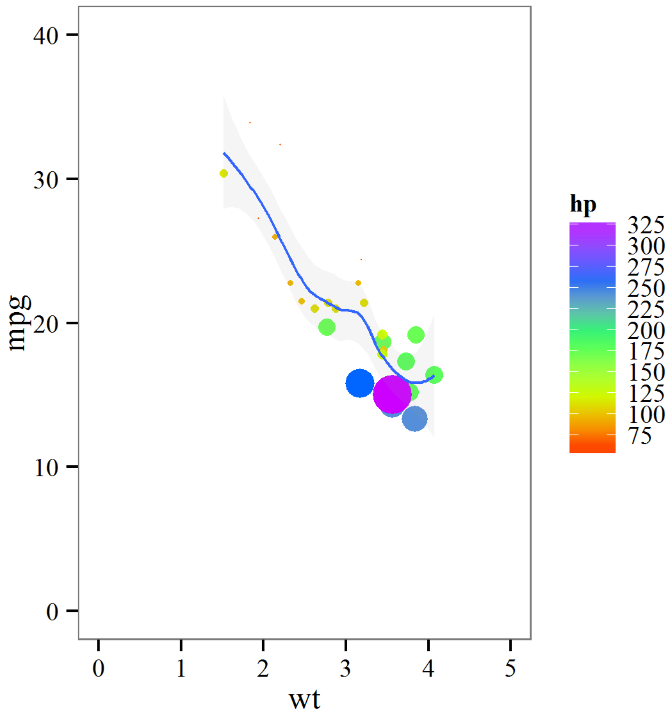


Figure 14: Change y-axis and x-axis tick marks.

A number of things have now changed (Figure 14), (1) the scales have reverted back to their original designations, (2) zero is now included on both axes, and (3) we have cut off data by constraining the axis at 5. One function of ggplot is that it only evaluates the data that is in the figure pane. So any smooths, etc., are informed only by the data that is seen. This can be changed to include the data in any smooth, but allow the figure to be "zoomed in" on an area of interest. To do this we use coord_cartesian() (Figure 15) .

```
ggplot(mtcars, aes(wt, mpg))+
  geom_point(aes(size=hp, color=hp))+
  geom_smooth(alpha=.1)+
  scale_colour_gradientn(colours=rainbow(5), breaks=as.vector(bs))+
  scale_size(range=c(0,7), breaks=as.vector(bs),guide = FALSE)+
  scale_y_continuous(breaks= seq(0,40,5), labels=c(0,"",10,"",20,"",30,"",40))+
  scale_x_continuous(breaks= seq(0,5,.5), labels=c(seq(0,2,.5),"","",3.5, "",4.5,""))+
  coord_cartesian(xlim=c(0,5),ylim=c(0,40))
```

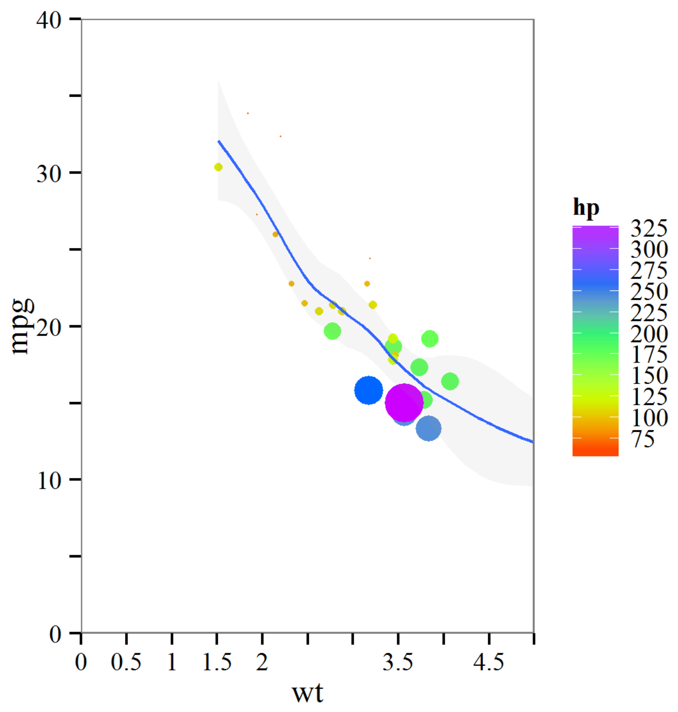


Figure 15: Combine legends.

The smooth now extends to the end of the pane as it indeed includes the values that are off the page. The points can be brought back into the figure by changing the x-axis limits (Figure 16.

```
ggplot(mtcars, aes(wt, mpg))+
  geom_point(aes(size=hp, color=hp))+
  geom_smooth(alpha=.1)+
  scale_colour_gradientn(colours=rainbow(5), breaks=as.vector(bs))+
  scale_size(range=c(0,7), breaks=as.vector(bs),guide = FALSE)+
  scale_y_continuous(breaks= seq(0,40,5), labels=c(0,"",10,"",20,"",30,"",40))+
  scale_x_continuous(breaks= seq(0,5,.5), labels=c(seq(0,2,.5),"","",3.5, "",4.5,""))+
  coord_cartesian(xlim=c(0,5.5),ylim=c(0,40))
```

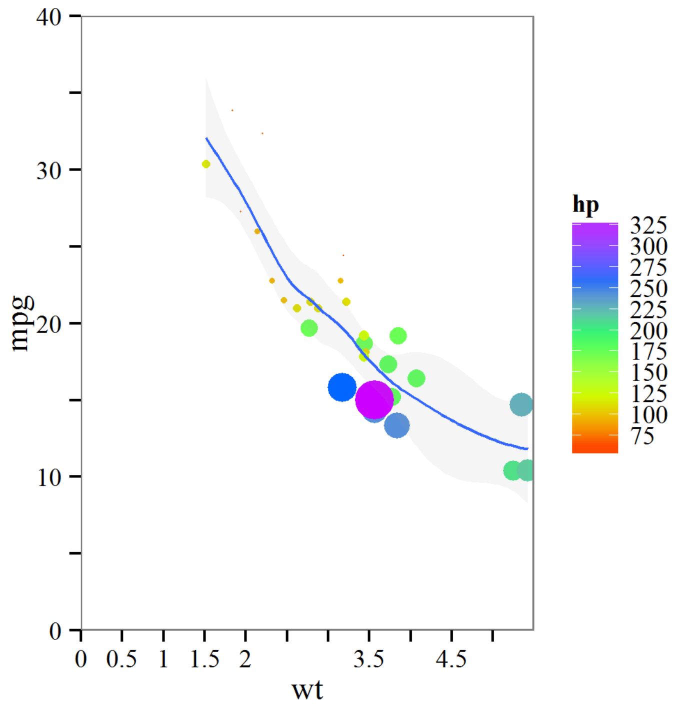


Figure 16: Combine legends.

5 Maps

5.1 Basic maps

There are a number of ways in R to plot maps and deal with raster images etc. I have no idea how to operate most of them. I do know a couple of easy ways to generate quick images though, they are presented below. The easiest method is to simply call a Google map location by name in ggmap. This needs an internet connection... For this example (Figure 17) I have called for a map of the Bering Sea, with a zoom of "4" (zoom needs to be an integer from 3 (continent) to 21 (building); more info here).

```
library(ggmap)
bs <- get_map(location = "Bering Sea", zoom = 4)
ggmap(bs)
```

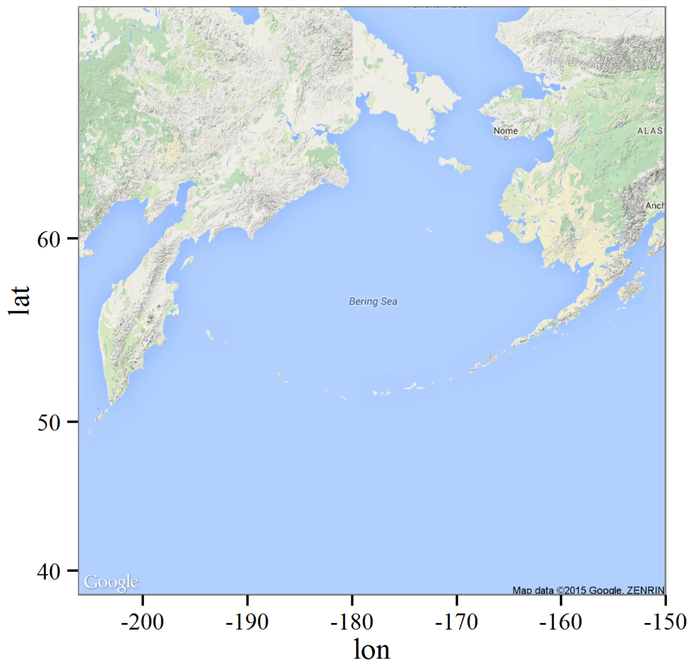


Figure 17: A map of the Bering Sea, pulled directly from Google Maps via ggmap.

A map can be adjusted (centered) on a specific location (Figure **??**), note that I changed the "zoom" on this map and used the longitude and latitude to determine the center of the Figure. Points can be added easily, in the same manner as any other ggplot object.

Note: For this example I simply made up some points to place in the general location and named them locs - you would have a separate dataframe with longitude and latitude positions in it.

First call the map.

```
bs1<-get_map(location = c(-170,63), zoom = 6)
```

Next add points to the map (Figure 18).

```
locs<-data.frame(long=c(-172,-173,-170,-170.5,-170.52),lat=c(62.1,62.2,62.3,63,65))

ggmap(bs1)+geom_point(data=locs,aes(long,lat),size=5,color=4)
```

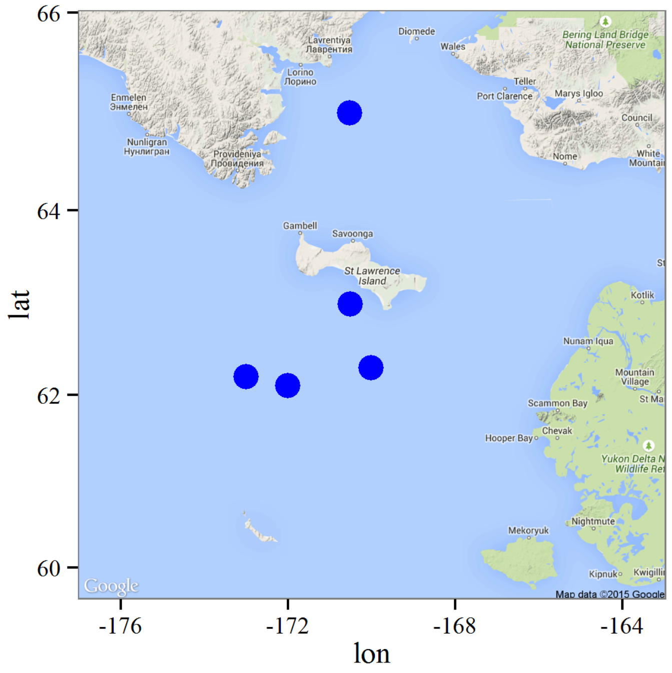


Figure 18: A slightly zoomed in map of the Bering Sea, centered on specific coordinate locations with randomly generated points overlayed.

Polygons can be easily added in a manner similar to adding point locations. Here I've simply made a polygon from the made-up database (Figure 19). You would have a separate database holding the location for your polygon(s).

```
ggmap(bs1)+geom_point(data=locs,aes(long,lat),size=5,color=4)+
    geom_polygon(aes(x = long, y = lat),data = locs,
    colour = NA, fill = "red", alpha = .5)
```

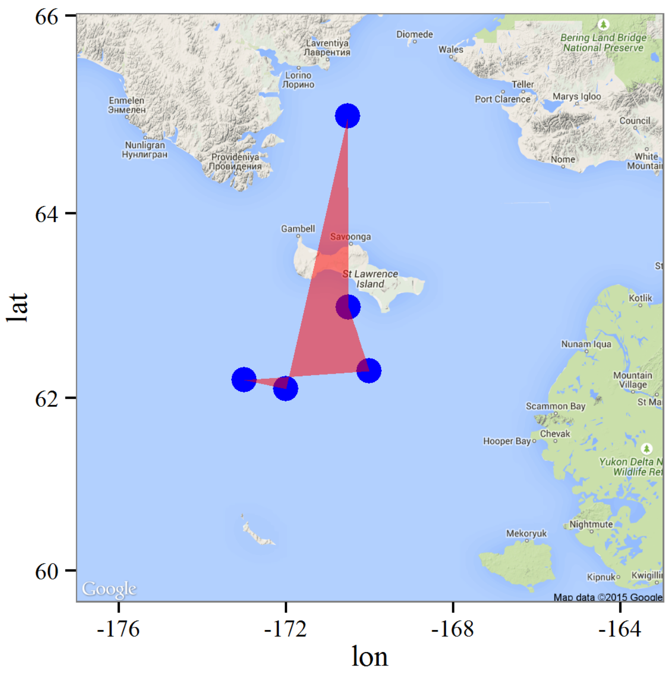


Figure 19: A slightly zoomed in map of the Bering Sea, centered on specific coordinate locations with a randomly generated polygon overlay.

5.2 Going deeper into the mapping realm (or the rabbit hole...)

Mapping can get very confusing very quickly depending upon what exactly you want to do, how you want to present information, etc. I am focusing on a limited subset of mapping, mainly generating maps for quick visualization of data for evaluating spatial components. There is a great deal of information on mapping using ggmap be sure to have a look. That said sometimes you just want a shaded map or outline map minus the labels that are included with the Google maps. This can be done as well! Lets generate a map of the western Gulf of Alaska.

```r
library(ggplot2)
library(maps)
library(mapdata)
library(mapproj)

ak<-map_data('worldHires','USA:Alaska')
ggplot()+geom_polygon(data=ak,aes(long,lat,group=group),fill=8,color="black")
```

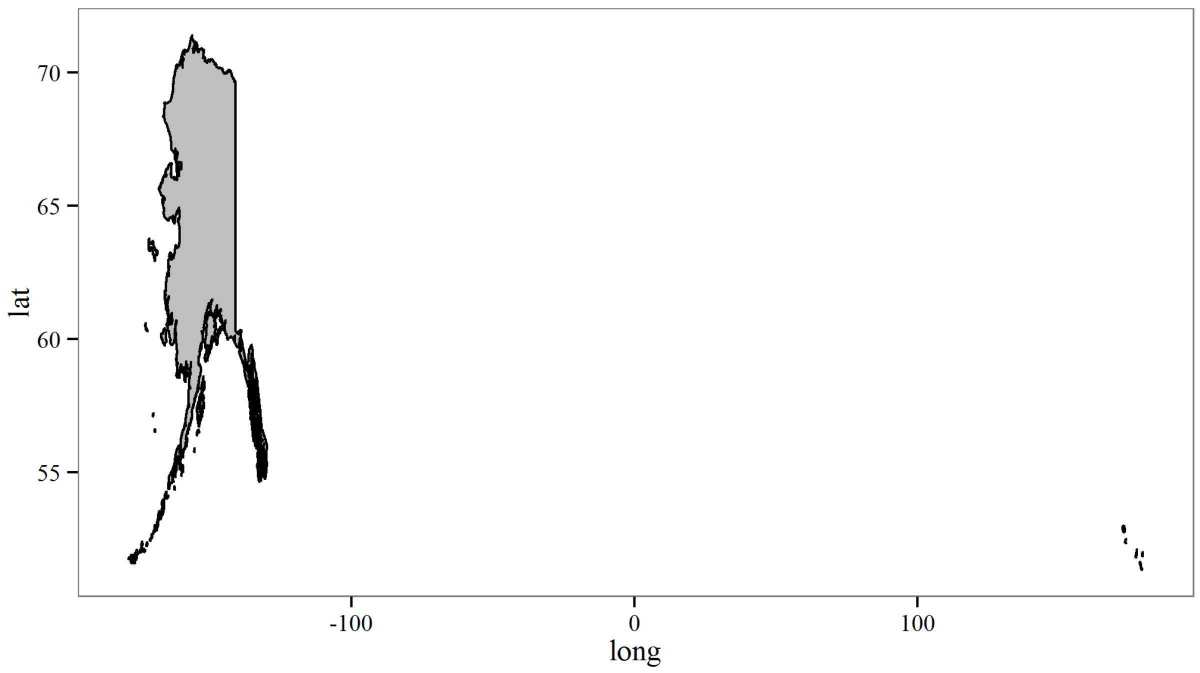


Figure 20: A map of Alaska, based on data from worldHires in the maps package.

Notice that this map doesn't seem quite right (Figure 20), there are two ways to deal with this one, if you don't need the far western Aleutian Islands you can simply constrain the plot (that is the method I will present); two, if you do need the Aleutian Islands then use “world2Hires” as opposed to 'worldHires' for the data source, this centers the map on the Pacific Ocean, but you will then have to deal with latitude and longitude adjustments (longitude is no longer negative ← small rabbit hole).

Lets return to the map we do want. There are two ways to constrain the data for the Aleutians (remove it, or simply constrain the limits of the plot). The first method here removes any points that are further west (east?) than 180^o. Then you can create a map with a light grey fill for the land and a black outline for the coast (Figure 21).

```
ak<-map_data('worldHires','USA:Alaska')
ak<-subset(ak,long<0) #drop the end of the Aleutian Islands, or use world2Hires
akmap<-ggplot()+geom_polygon(data=ak,aes(long,lat,group=group),fill=8,color="black")
akmap
```

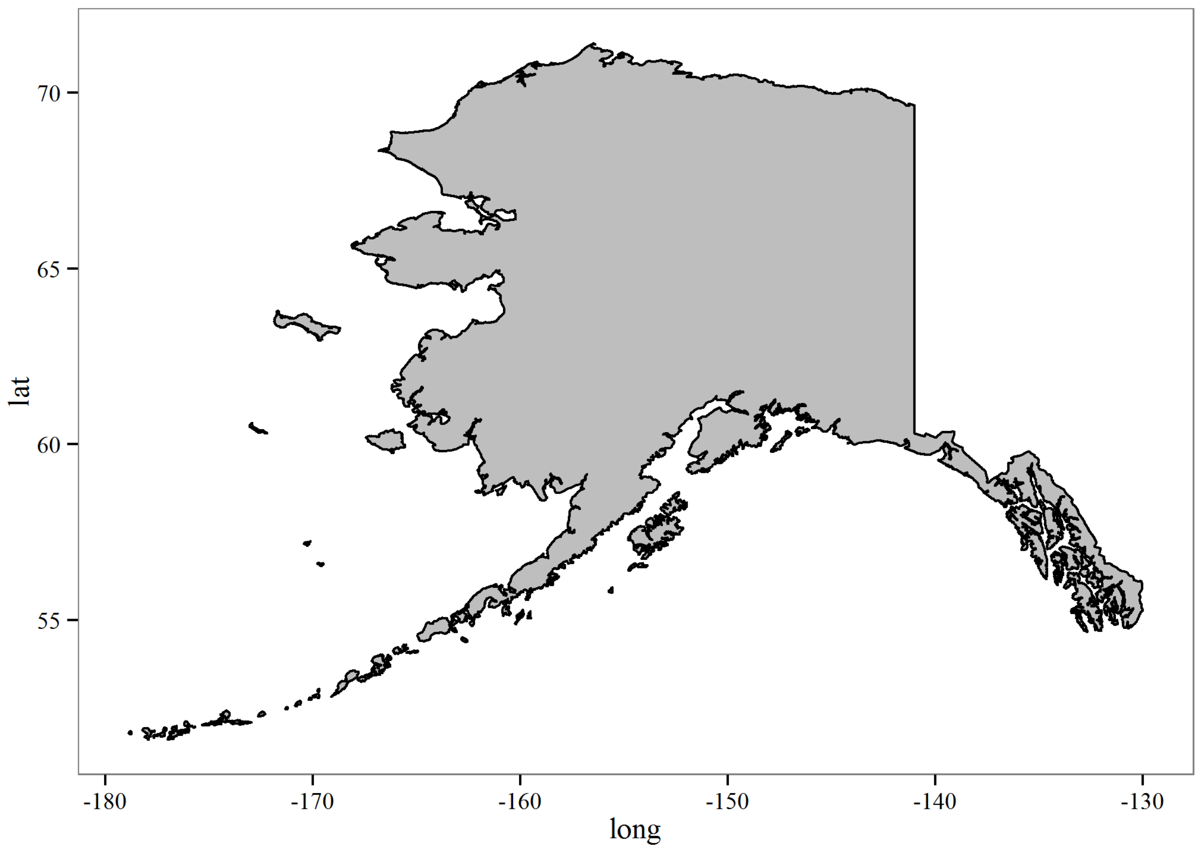


Figure 21: ggplot2 map of Alaska based upon worldHires data.

Or we can simply constrain the data for the map if we are only interested in a particular area of Alaska. We can also clean the map up while zooming into a particular areas (Figure 22), in this case the western Gulf of Alaska. Note that the code to zoom in is not simply done by changing the x and y limits. When you constrain the x and y limits in ggplot it throws out any data outside of the image, therefore the polygon shapes ("groups") will not be complete and the image will not be pretty. In this plot I've incorporated the limits in a projection wrapper (coord_map), there are a number of different projections that you can use though this will lead to a BIG rabbit hole! I've added a background color (aliceblue) as well as incorporated degree symbols into the x- and y-axis labels. Data points and polygons can now be added as additional layers as shown earlier.

```
ak<-map_data('worldHires','USA:Alaska')
ggplot()+geom_polygon(data=ak,aes(long,lat,group=group),fill=8,color="black")+
    theme(panel.background=element_rect(fill='aliceblue'))+
    xlab(expression(paste(Longitude^o,~'W')))+
    ylab(expression(paste(Latitude^o,~'N')))+
    coord_map(xlim = c(-165, -145),ylim = c(54, 61))
```

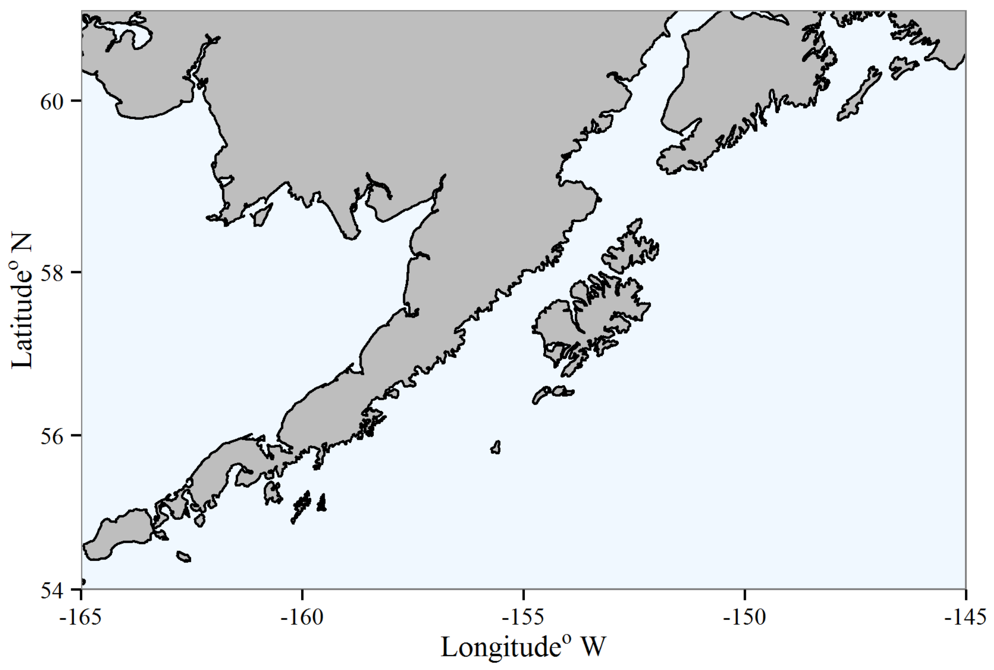


Figure 22: ggplot2 map of Alaska based upon worldHires data all dressed up with color.

Figure 23 is the same map without color (better for reports, etc.).

```
ggplot()+geom_polygon(data=ak,aes(long,lat,group=group),fill=8,color='black') +
   theme(panel.background=element_rect(fill='white')) +
   xlab(expression(paste(Longitude^o,~'W'))) +
   ylab(expression(paste(Latitude^o,~'W')))+
    coord_map(xlim = c(-165, -145),ylim = c(54, 61))
```

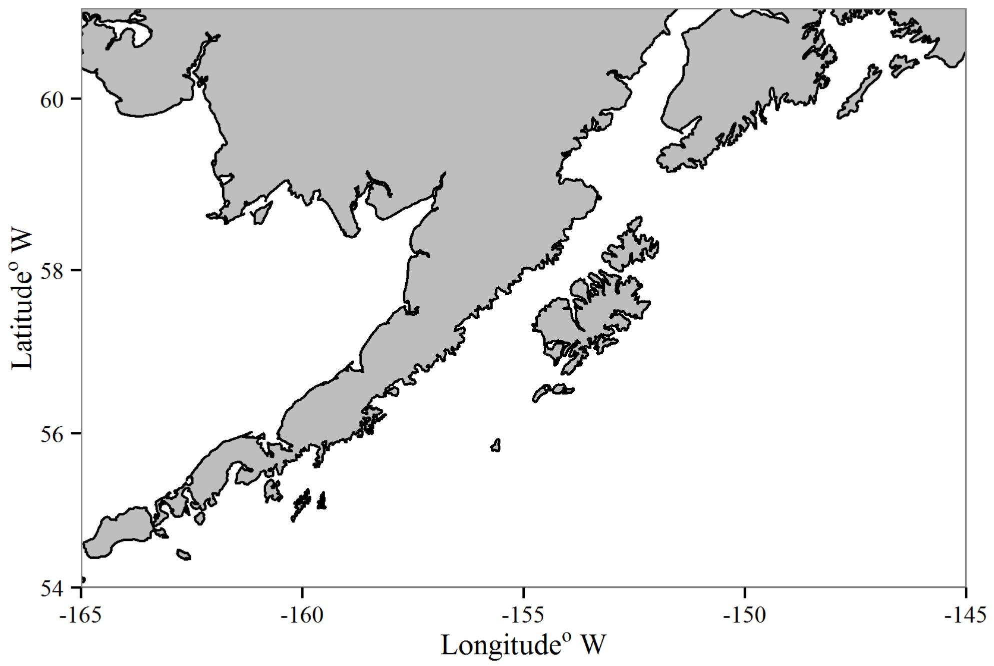


Figure 23: Zoomed in ggplot map of the western Gulf of Alaska based upon worldHires data.