Project #3 stat319

Alan Lin

4/8/2022

1a. How many of the countries in the CIA Factbook don't have an ISO 3166 code?

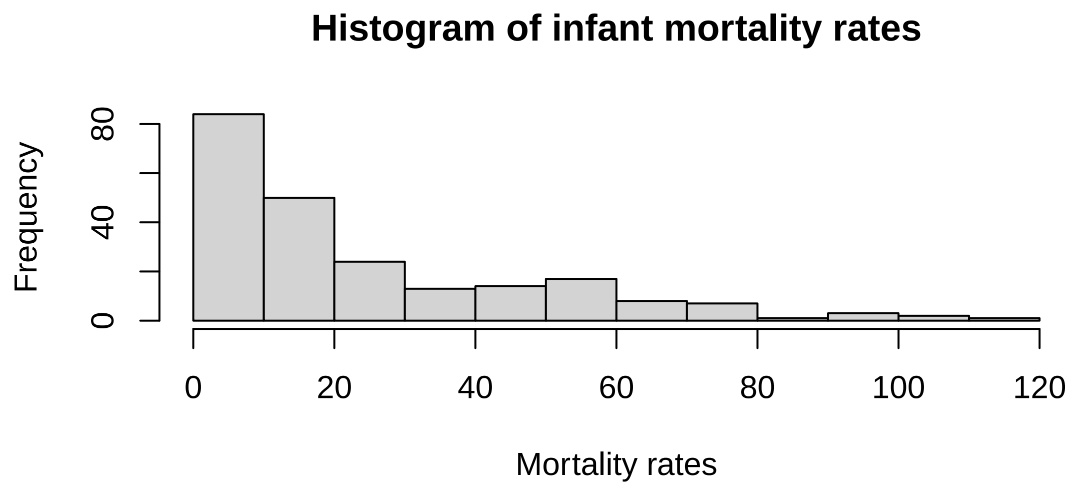
We can easily find the number of countries in the dataframe that have "-" as a value for ISO-3166. There are 28 countries that don't have an ISO 3166 code.

1b. How many of these "countries" don't have a CIA Factbook 2-letter country abbreviation, either?

We can also quickly find this by adding a second condition: is there a "-" for the cia column in the dataframe, as well as having "-" for ISO-3166? There are 6 countries that fulfill these two conditions.

2a. Create a histogram of infant mortality rates, and describe its shape.

The histogram is heavily right skewed, with most of the countries having low infant mortality rates. Since it is so skewed, the median is a better estimate for the center of the data. The median infant mortality rate is probably around 15 deaths/1000 live births a year.



3. Describe your process of finding and cleaning this data.

After a quick Google search, this data was taken from a GitHub folder at https://gist.github.com/tadast/8827699 that contained countries with their ISO3166 Alpha-2 code, Alpha-3 code, UN M49, average latitude, and average longitude. The original column names weren't very programming-friendly, so after using dplyr to select for the three columns that I wanted - the ISO3166 Alpha-2 code, average latitude, and average longitude - I renamed the column names to "iso3166", "latitude", and "longitude".

4. Merging 4 datatables together. Describe any issues that come up and how it was handled.

The geographical dataframe had isocodes that had an empty character right before the actual code, so when merging, it wasn't actually matching it. Therefore, I needed to take out the whitespaces first before joining. I achieved this by using the gsub function in base R. Also, I used the dataframe with the most rows as the left table in each left join, which was the cross-reference list of country data codes, since left joins keeps all rows even if there are no matches. dplyr would automatically assign a "NA" value when there is no match.

5. Find the mean mortality rate for all countries with population less than 10 million, and for those countries with population greater than 50 million.

By subsetting the merged dataset for all countries with population value less than 10 million or more than 50 million, I could then use the mean function and the na.rm = T option to find the mean mortality rate.

This turns out to be 18.8623 deaths/1000 live births for countries with population less than 10 million and 25.786 for countries with population greater than 50 million.

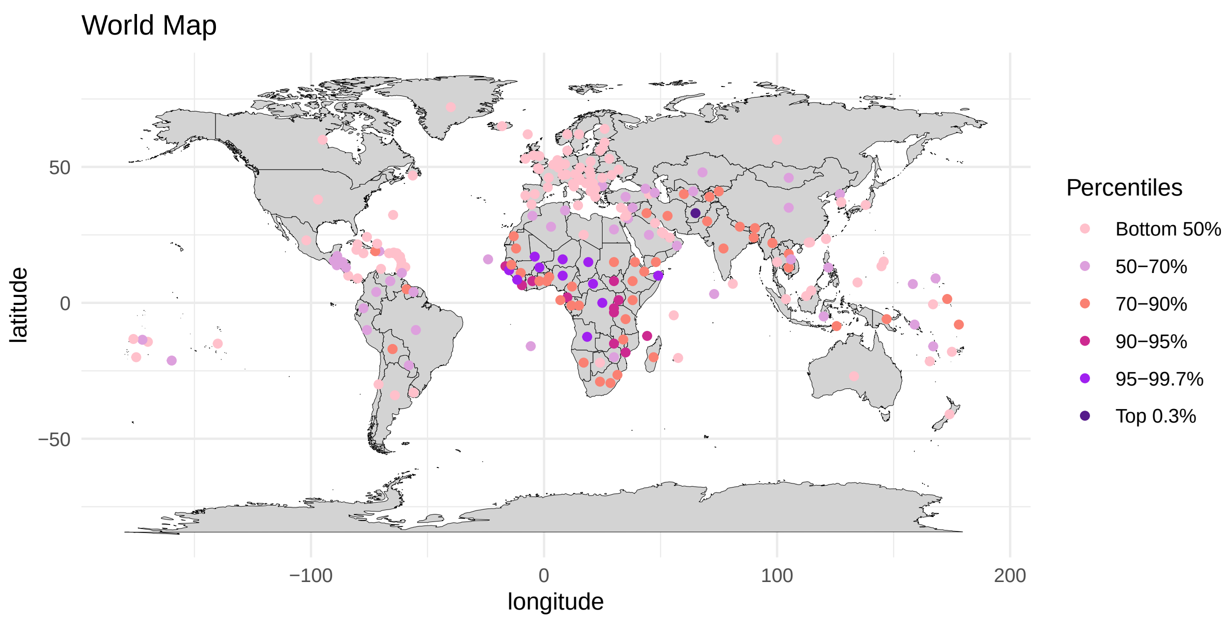
6. Create a table showing how many countries fall into each level of your discretized mortality rates. Also explain how and why you selected the intervals to define the factor levels.

There are 117 countries in the bottom 50% of the world's mortality rates. They make up a large percentage of the total number of countries in the dataset. The next group spans from the 50th percentile to the 70th percentile, which has 47 countries. The 70th to the 90th percentile has 46 countries. Now, even though these aren't confidence intervals, I thought it would be entertaining to look at commonly used confidence intervals for statistical inference such as 90% and 95%. I chose to look at the 99.7th percentile just because of the CLT rule for 3 standard deviations away from the sample mean. 12 countries are in the top 90th-95th percentile, 11 countries are in the 95th-99.7th percentile, and 1 country is in the top 0.3% of mortality rates.

```
## cut
## Bottom 50%     50-70%     70-90%     90-95%   95-99.7%   Top 0.3%
##        116         47         46         12         11          1
```

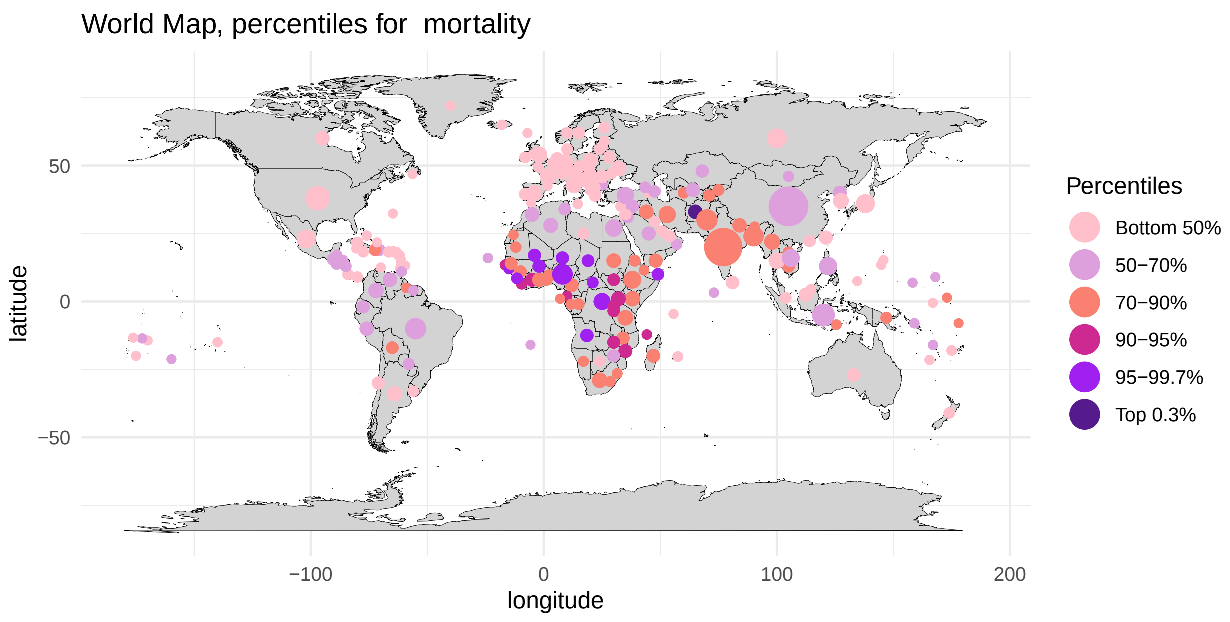
7. Create a world map, with colored circles showing infant mortality rates. Include a legend showing that colors correspond to what intervals - or quantiles - of the mortality rates.

The world map uses the infant mortality data, creates a column that has the colors attributed to each group as designated from before. The colors go from a light pink to a dark purple. That means the lighter the color, the smaller the infant mortality rate is. As we can see, most of Europe (pretty developed countries) have light pink circles, while third-world countries have comparatively higher infant mortality rates, with only country, Afghanistan having the highest infant mortality rate, being at the top 0.3% in the world.



8. Create another map, this time where the area of each colored circles corresponds to the population of the corresponding ccountry.

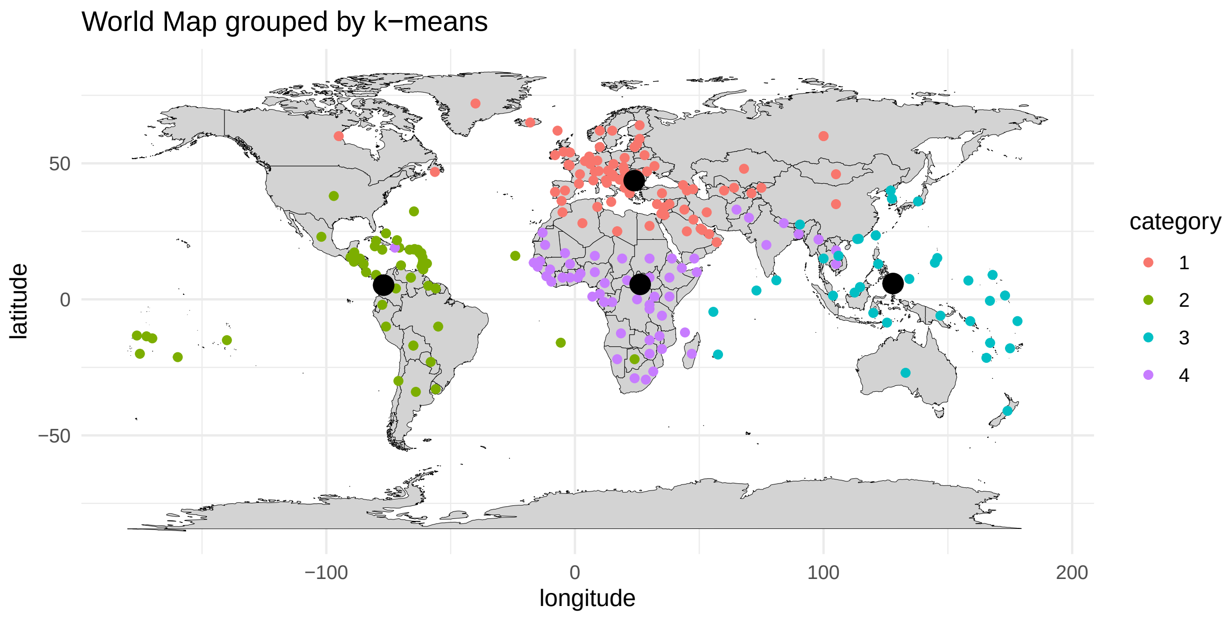
This accurately reflects the population size of each country, with China, India, and the US being the largest circles, as expected. The data still reflects the infant mortality rate, with more developed countries having lower mortality rates.



9. Include the code of your kmeans() function in the Appendix of your report. Then apply your k-means clustering algorithm to the three variables: latitude, longitude, infant mortality. Create a map with an equal sized circle on each country, where the circle color represents the k-means group classification.

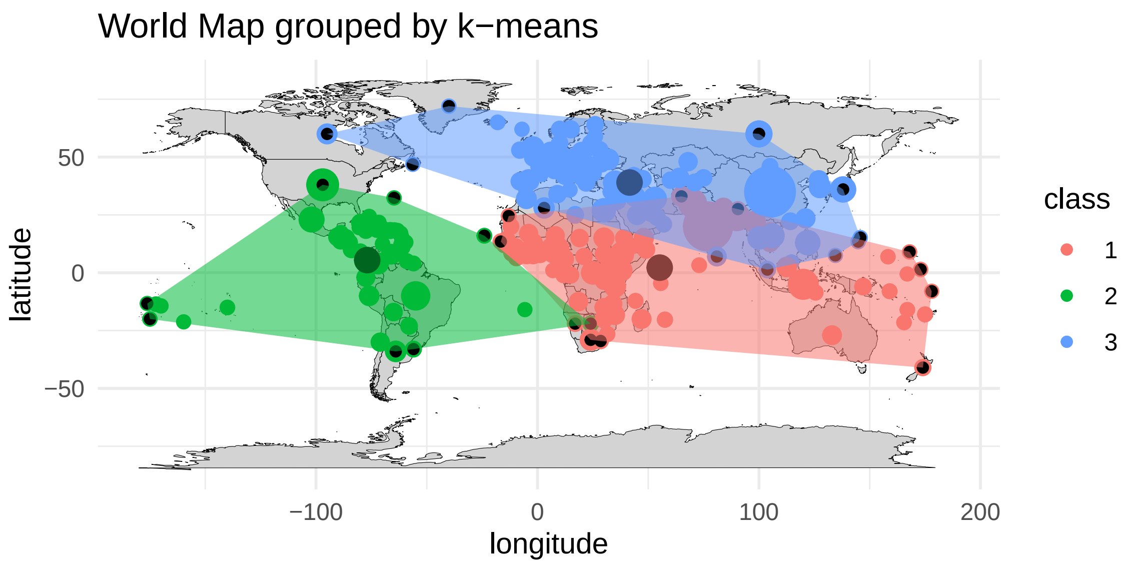
The k-means algorithm works by taking in a k-value for the number of classifications that we want, as well as a matrix. We should have removed any rows in the matrix that has incomplete data (NA values) because that would mess up the Euclidean distance calculations and calculation of the mean for the new centroids.

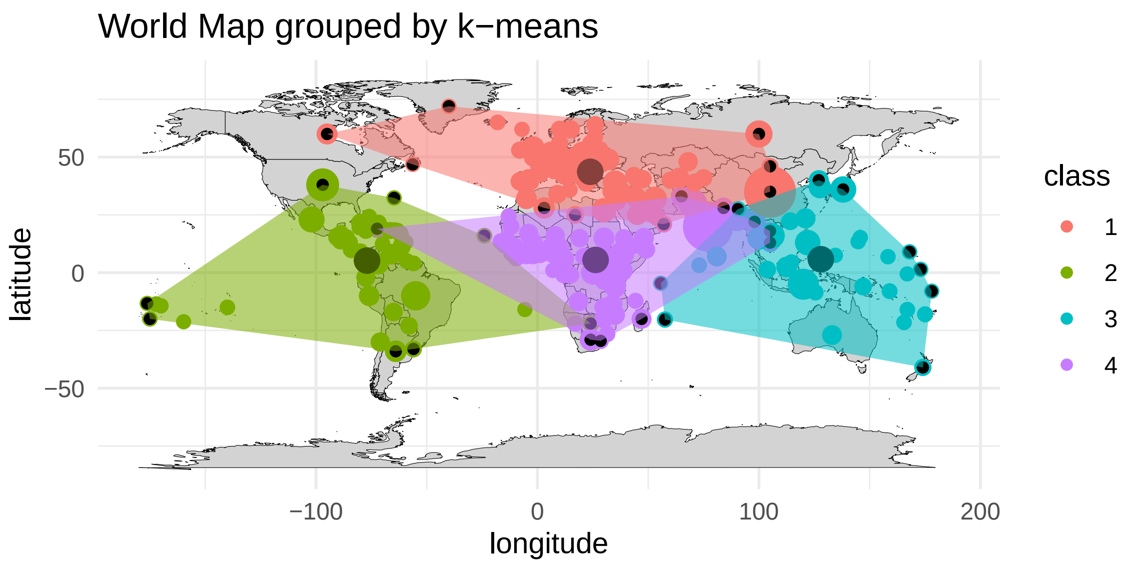
On the map, the black dots represent the centroids at the end of the k-means clustering algorithm. The countries seem to cluster pretty well, with one group in the Pacific Islands/Americas, one in Europe/North Africa, and the fourth group in Asia/Oceania.

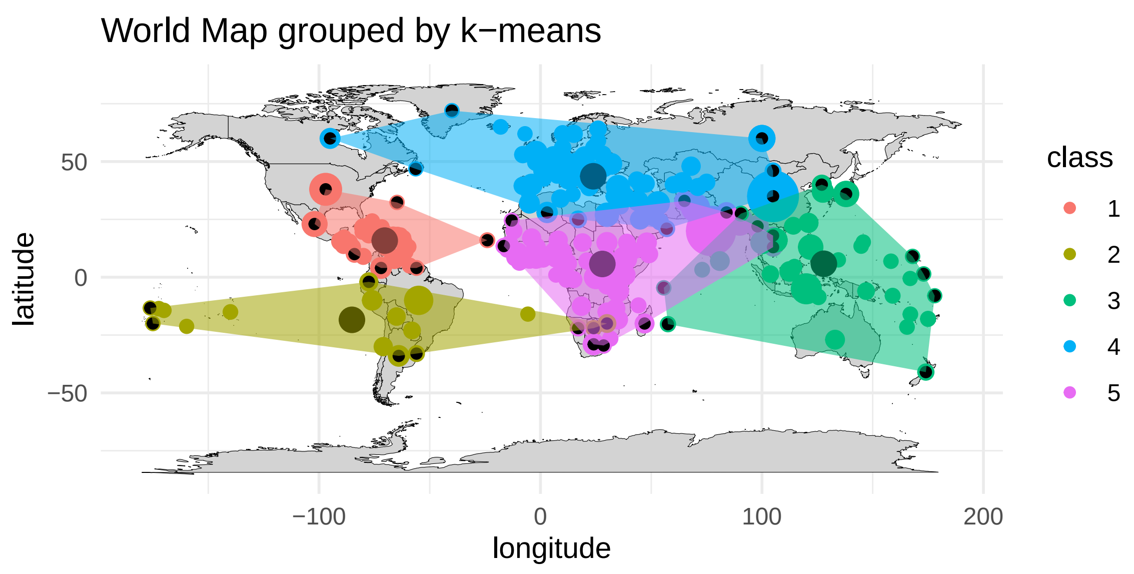


10. Execute your regionalMap() function for several different values of k, and include the resulting maps.

While the groupings are overlapping with one another, the polygons created are indeed the convex hulls of each group. Since the groupings aren't based solely on geographical location(latitude and longitude), that means that the countries in the overlapping groups are more closely related to other countries in their respective group based on mortality rate. Still, the overwhelming majority of the grouping seems to be based on geographical location, rather than their mortality rates.


World Map grouped by k−means
latitude
50
0
−50
−100
0
100
200
longitude
class
1
2
3


World Map grouped by k−means
latitude
50
0
−50
−100
0
100
200
longitude
class
1
2
3
4



```
## Time difference of 2.850174 secs
```

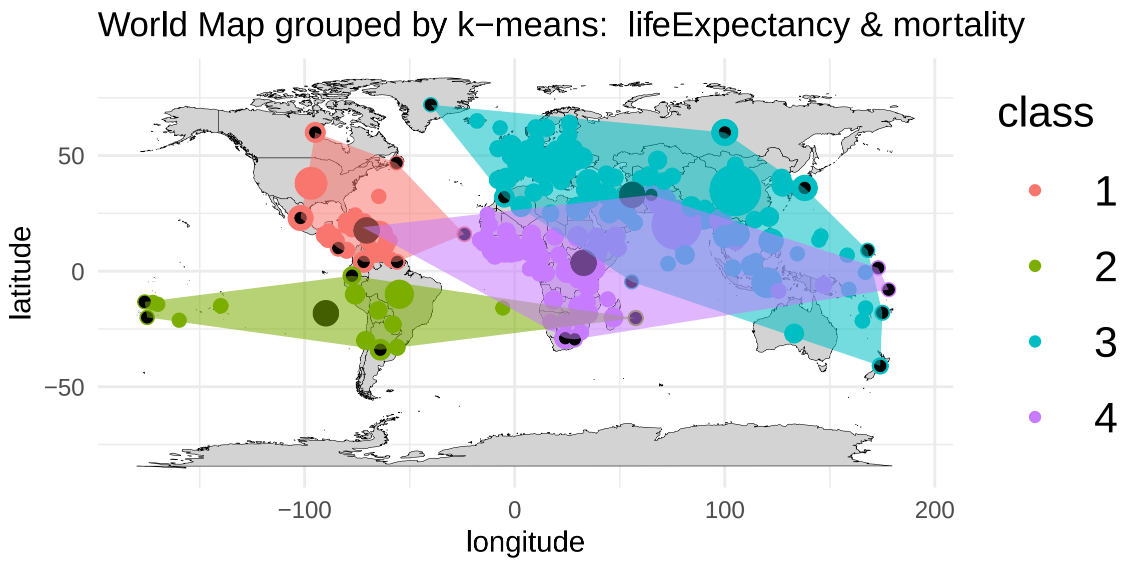
Extensions to the project.

First, although it wasn't very intensive, I added another parameter to the kmeans function in order to return the centroids at the end of the algorithm. This allows me to specify when I want the matrix of centroids in order to add the resulting centroid for each group onto the world map. using the returned matrix of centroids, we have to unscale the latitude and longitudes, so by just saving the center as well as the standard deviation from the attributes in the scale function, we can quickly figure out the respective latitude and longitudes of the centroids. It's easy to see where these centroids are because they are clearly marked out with a color different from the groups. Again, it should be mentioned that these centroids are only geographically placed, and not entirely indicative of the clustered group. That means that although there are several points that are geographically closer to the centroids of another group, it is the mortality rate that is keeping the country in another group.

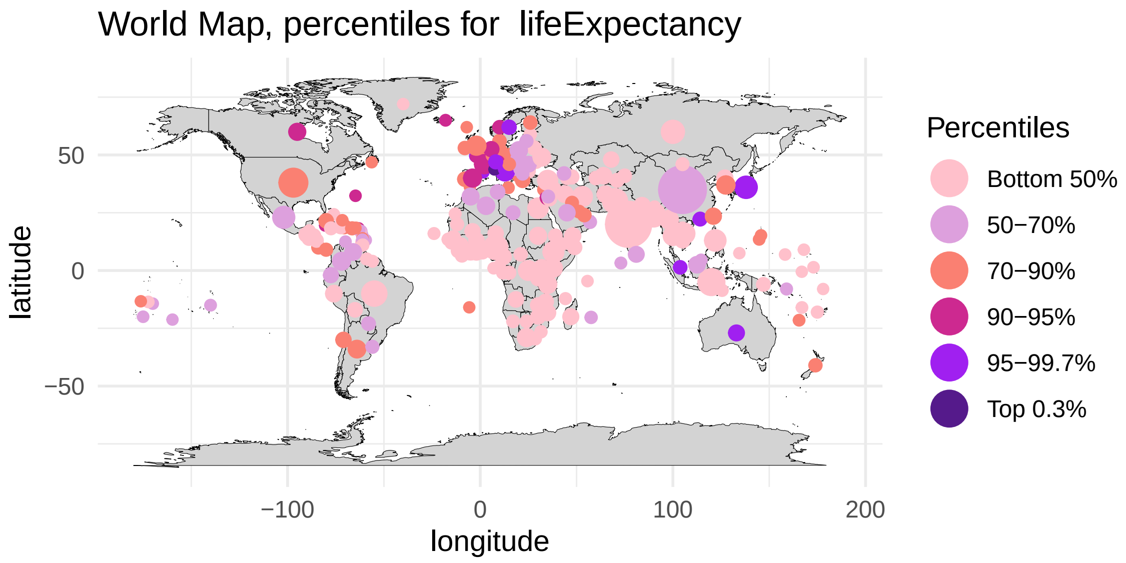
Another extension I did was to figure out how to use ggplot instead of base R to draw the world maps and the polygons to depict the convex hulls of each group. This took a while to figure out, but because ggplot cannot continuously overlay like base R can, I had to figure out how to create a data frame that houses the longitudes, latitudes, and group assignment for each point in the convex hull. After that, it was a matter of plotting the points to the map using the separate data frame for the polygons. I have to say, there was a lot of fidgeting with the amount of lapply, sapply, and unlisting but because I avoided using for loops, the vectorized functions allow me to generate regional maps with relative speed, at around 1 second for each function call. See the appendix for the new updated regionalMap. The baseR is also included in the appendix.

I looked through the CIA Factbook for other demographics of interest and chose to look at life expectancy at birth. Essentially, all I'm doing is grabbing the values from the XML sheet and then left joining to the master merge data, in order to create a large dataframe with multiple demographics of interest. I could then tweak my regionalMap function to take in one or multiple variables of interest as a vector. For example, if I wanted to do k-means for two demographics of interest, like mortality and life expectancy at birth, I would pass in those two variables in a vector to the function. Of course, we'll still use population to size the circles on the map, and the longitude/latitudes of each country, so we simply grab a subset of the merge dataframe of those variables to do the k-means assignments.

Here, we have an example of using two variables of interest to do k-means sorting. As before, most of these are arranged in clusters geographically, but something about either life expectancy or mortality is making it much more similar to its assignment than countries that are closer in vicinity. In the example below, many of the clusters overlap and have huge ranges, such as the one from the Pacific Islands to Africa, and the Americas to Oceania.



Additionally, I wrote a function called percentiles which takes in a variable of interest, calculates the percentiles (see Deliverable 8), and then displays the labels for each country depending on which group they fall into. This helps us visualize geographically how some demographics of interest are in relation to one another. For example, we already saw that mortality rate is higher in third-world countries, so their percentiles are at the higher end of the spectrum, but if we look at life expectancy at birth, then we see third-world countries, especially in Africa are at the bottom of the spectrum (life expectancy is low in these countries compared to developed countries in Europe).



I was also interested in creating a Shiny app that will display dynamic plots, similar to ones generated throughout this project, which can be viewed at this link:

https://alan-lin-stat-projects.shinyapps.io/project3-stat319/.

I allowed users to change the seed in order to select different centroids, while allowing them to change the number of groups desired from the k-means algorithm. Again, the time it takes to generate these maps are pretty quickly, usually rendering after a second or two. I also wanted to allow users to change the demographic of interest, rather than just the mortality rate. Here, we allow for up to three variables of interest to be chosen. Because it's much easier to select variables in a Shiny app, I've compiled many of the variables in the CIA factbook for use in the Shiny app, including: mortality, lifeExpectancy, healthSpending, fertilityRate, obesity, and many more. This is possible by just reusing the regionalMap1 function, which takes in a vector of variables (stored from Shiny input) and outputting the world map into the app. I also created a tab along the top to click from to allow users to explore the percentiles/histograms of a variable of interest. Finally, by clicking the three bars at the top of the page, there is a separate page that acts as the glossary for the variables of interest, in case any user wants to know what their working with.

As is the case with most of these maps, many of the variables (regardless of number of variables chosen) resulted in a clustering that seemed to be based on geographical location. Of course, there are exceptions to the groups which would be due to the demographic of interest.

Appendix

K-means function, slightly changed to not only take a data frame and the number of groups, but also a boolean to return the centroids at the end of the clustering.

```
kmeans <- function(k, x, returnCent = FALSE){



  p <- ncol(x)  ## get # col
  n <- nrow(x)  ## get # row

  stand <- matrix(as.numeric(scale(x)), n, p) ## standardized matrix
  #stand <- as.matrix(x)



  rows <- sample(nrow(stand), k, replace = F) ## randomly select

  centroids <- stand[rows,]  ## initialize the centroids


  while (TRUE){

    groups <- c() ## initialize assignments

    for (i in 1:n){


      g <- which.min(apply(apply(centroids, 1, function(x){
        (stand[i,] - x)^2

      }), 2, function(x){sqrt(sum(x))})) ### find the closest centroid in Euclidean distance

      groups <- c(groups, g) ## append to growing list of classifications


    }

    oldcentroids <- centroids ## save a copy of the old centroids
    centroids <- matrix(c(sapply(1:k, function(x) {
      l <- stand[which(groups == x),]
      apply(l, 2, mean)
    })), ncol = p, nrow = k, byrow = T)

    ## calculate the mean of the new centroids
    ## compare to the old centroid.

    ## if identical, that means the assignments were the same in current iteration and the previous, so
    if(identical(oldcentroids, centroids)) break


  }

    ## return the centroids if prompted to

   if(returnCent) return(centroids)

    ## otherwise return the assignemnts.
   return(groups)
}
```

Old function for Deliverable 10 using base R

```
regionalMap <- function(k){

      ## keep centroids and assignments the same
      seed <- sample(1:10000, 1)


      a <- merge[,4:7] ## merge has the entire dataframe,
      # this is just grabbing the 3 columns needed : population, lat, long, and mortality rate

      ## find the complete cases, no NA's
      a <- a[complete.cases(a),]


      ## need these two to "unscale" centroids in order to plot them
      center <- attributes(scale(a[,2:4]))$'scaled:center'
      scale <- attributes(scale(a[,2:4]))$'scaled:scale'





      ## get assignments from kmeans
      set.seed(seed)
      class <- kmeans(k, a[,2:4])


      ## get centroids from kmeans
      set.seed(seed)
      cents <- kmeans(k,a[,2:4], TRUE)
      a$class <- class

      op <- par(cex = 1)
      area <- sqrt(a$population)
      f <- scale(area, F, T) + 2

      long <- a$longitude
      lat <- a$latitude

      clat <- cents[,2] * scale[2] + center[2]
      clong <- cents[,3] * scale[3] + center[3]

      col <- rainbow(k, alpha = 0.2)

      a$color <- 0

      for(i in 1:k){
        a[which(a$class == i), "color"] <- col[i]

      }

      c <- a$color
      l <- 1:k


      map("world")
      symbols(x = long, y = lat, add = T, inches = F, circles = f, fg = c, bg = c)
      legend(x = -180, y = 80, legend = l, col = col, pch = 19, title = "Classification")
      symbols(x = clong, y = clat, add = T, inches = F, circles = rep(3, k), fg = "black", bg= "black")


      for(i in 1:k){
        l <- a[which(class == i),]
        ind <- chull(l$longitude,l$latitude)
        hull_x <- l[ind,"longitude"]
        hull_y <- l[ind,"latitude"]
        points(hull_x, hull_y, pch = 19, col = 'black')


        polygon(hull_x, hull_y, border = 'black', col = col[i])

      }



}
```

updated

```
regionalMap1 <- function(k, interest){

  getCol <- c("population", "latitude","longitude",interest)
```

```r
a <- merge[,getCol]
a <- a[complete.cases(a),]



center <- attributes(scale(a[-1]))$'scaled:center'
scale <- attributes(scale(a[-1]))$'scaled:scale'


set.seed(10)
class <- kmeans(k, a[-1])


set.seed(10)
cents <- kmeans(k,a[-1], TRUE)

a$class <- as.factor(class)

op <- par(cex = 0.7)
area <- sqrt(a$population)
f <- scale(area, F, T) + 2

long <- a$longitude
lat <- a$latitude

clat <- cents[,1] * scale[1] + center[1]
clong <- cents[,2] * scale[2] + center[2]

if(length(interest) > 1) {
  t <- paste(interest, collapse = " & ")
  map <- baseMap +
  geom_point(data = a,
           aes(x = longitude, y = latitude, color = class), size = f) +
  geom_point(aes(x = clong, y = clat), color = "black", size = 4) +
  labs(x = "longitude",
     y = "latitude",
     title = paste("World Map grouped by k-means: ", t))

} else {


map <- baseMap +
geom_point(data = a,
           aes(x = longitude, y = latitude, color = class), size = f) +
geom_point(aes(x = clong, y = clat), color = "black", size = 4) +
labs(x = "longitude",
     y = "latitude",
     title = paste("World Map grouped by k-means: ", interest))

}

hull <- lapply(1:k, function(x){
  l <- a[which(class == x),]
```

```
    ind <- chull(l$longitude,l$latitude)
    hull_x <- l[ind,"longitude"]
    hull_y <- l[ind,"latitude"]
    return(list(hull_x=hull_x, hull_y = hull_y))
  })


   testhull <- sapply(hull, "[[", 'hull_x')
   hullx <- unlist(sapply(hull, "[[", 'hull_x'))
   hully <- unlist(sapply(hull, "[[", 'hull_y'))

   colors <- sapply(1:k, function(x) length(testhull[[x]]))
   colors1 <- unlist(sapply(1:k, function(x) rep(x,colors[x])))

   polymap <- data.frame(hullx = hullx, hully = hully, colors = colors1)


   map +
     geom_point(data = polymap,
                aes(x = hullx, y = hully), show.legend= F) +
     geom_polygon(data = polymap,
                aes(x=  hullx, y = hully, fill = factor(colors),
                    alpha = 0.2)) + guides(alpha = "none", fill = "none")  +
     theme(legend.text = element_text(size = 16),
          legend.title = element_text(size = 16),
          legend.key.height = unit(1, 'cm'),
          legend.key.width = unit(1,'cm'))

}
```