NASA Data Processing

October 21, 2015

Setting up the workspace

```
rm(list = ls())
setwd("~/Desktop/STA_141/assignment_2/")

require(scales) # for changing alpha level in plots
require(png)
require(grid)
require(magrittr) # for updating row names (while returning object)
require(maps)
require(lattice)
```

Step 1

Our general approach for cleaning the data is as follows. First, identify the row with "TIME" and extract the date string, which will then be converted to a date type object. Then, take the next line which contains longitude values with E/W designations and convert them to +/- numeric values. We'll then skip a line, and read in the rest of the data in the file, adding our newly cleaned longitude values as column names (with some slight alterations since the first three columns aren't readings). We'll extract the first column and clean up the latitude values in a similar manner to the longitudes and keep these latitudes stored for later. Finally we'll stack the data frame, column by column and repeat the latitudes values so that they will align with their original orientation and add the date.

```
##########
# Step 1 #
##########

fix_lat_long =
    # INPUT:
    #   - lat_long: character vector of latitudes or longitudes
    #   - long_TF: TRUE/FALSE for longitude/latitude
    #
    # OUTPUT: numeric vector with fixed lat/long values
    #
    # DOC:
    #   - correct lat/long from N/S/E/W to numeric with corresponding +/- sign
    #   - from: http://itouchmap.com/latlong.html
    #   - Use: + for N Lat or E Long; - for S Lat or W Long.
function(lat_long, long_TF = TRUE)
{
    # different instructions for longitude (E/W) and latitude (N/S)
    if (long_TF) {
        direction = grepl("W$", lat_long)
        reg = "[WE]"
    } else {
        direction = grepl("S$", lat_long)
        reg = "[NS]"
    }
```

```
    # replace S and N (or W and E) with -1 and +1 respectfully
    direction = c(1, -1)[ direction + 1]

    # update lat_long with the correct sign
    lat_long = as.numeric( gsub(reg, "", lat_long) ) * direction

    lat_long
}

get_date =
    # INPUT: header: a character vector, the top portion of a file
    # OUTPUT: a date object of class Date
    # DOC: extract the date as date class from row with TIME
function(text)
{
    time_line = which( grepl("TIME", text) )
    time = text[ time_line ]
    time = strsplit(time, " +")[[1]]

    # the second to last item is the date
    date = time[ length(time) - 1 ]
    date = as.Date(date, "%d-%b-%Y")

    date
}

clean_data =
    # INPUT:
    #   - file: the name of a file
    #   - directory: path to the file
    #
    # OUTPUT: a data frame with columns: date, lat, long, variable
    # DOC: cleans the data and returns a data frame with 4 columns
function(file, directory) {

    # get the variable name from the file name
    variable = strsplit(file, "[0-9]")[[1]][1]

    # build full path to the file and read in 7 lines
    file = paste0(directory, "/", file)
    header = readLines(file, 7)

    # extract the date, raise exception if date is NA
    date = get_date(header)
    stopifnot( !is.na(date) )

    # line with longitude follows the TIME line, split on white space
    longitude = header[ which( grepl("TIME", header) ) + 1 ]
    longitude = strsplit(longitude, " +")[[1]][-1]
    longitude = fix_lat_long(longitude)

    # getting the classes of each column, passing to read.table
    classes = c( rep("character", 3), sapply(longitude, class) )
```

```
    # we found several "...." values which we'll take to be NA,
    # also setting column classes to be numeric to potentially find other NAs
    df = read.table(file, sep = "", skip = 7, na.strings = "....", colClasses = classes)

    # extracting and fixing latitudes
    latitude = as.character( df[, 1] )
    latitude = fix_lat_long(latitude, long = FALSE)

    # remove first three columns
    df = df[, -c(1,2,3)]
    names(df) = longitude

    # stack data frame and add latitude and date columns
    df = stack(df)
    df$lat = rep( latitude, length(longitude) )
    df$date = date

    # rename columns
    names(df) = c(variable, "long", "lat", "date")
    df$long = as.numeric( as.character( df$long ) )

    # reorder columns
    df = df[, c(4, 3, 2, 1)]

    df
}

# download and upzip data
filename = "nasa.zip"
download.file("http://eeyore.ucdavis.edu/stat141/Data/nasa.zip", filename)
unzip(filename)

# get the names of all files in the unziped file
directory = "./NASA"
files = list.files(directory)

# remove .dat files from vector
files = files[ !grepl(".dat", files) ]

# add variable names to each file
names_for_files = sapply( strsplit(files, "[0-9]"), function(x) x[1] )
names(files) = names_for_files

# clean the data in each file then split by variable name
list_of_dfs = lapply(files, clean_data, directory = directory)
list_of_dfs = split(list_of_dfs, names(list_of_dfs))

combine_dfs = function(df_list) {
    set_rownames( do.call("rbind", df_list), NULL)
}

# combine all data frames with the same variable
df_list = lapply( list_of_dfs, combine_dfs )
```

```
# lapply(df_list, head, 3)
sapply( df_list, dim)
```

```
##      cloudhigh cloudlow cloudmid ozone pressure surftemp temperature
## [1,]     41472    41472    41472 41472    41472    41472       41472
## [2,]         4        4        4     4        4        4           4
```

Step 2

In order to combine the seven data frames, we need to make sure the date, latitude, and longitude in row i of data frame 1, is the same date, latitude, and longitude in row i for all six other data frames. If we let df_j represent the data frame for variable j, then we'll simply make sure df_1 and df_k for $k = 2, \ldots, 7$ are identical in terms of date, latitude, and longitude.

```
##########
# Step 2 #
##########

check_same =
    # INPUT: list of data frames and a column name for comparison
    # OUTPUT: returns TRUE if all col_names in each df are identical, FALSE otherwise
function(list_of_dfs, col_name)
{
    # extract vector from first df for comparison
    first_vector = list_of_dfs[[1]][col_name]

    # check equality between and first df and the remaining dfs
    equal_cols = sapply(list_of_dfs, function(df) all( df[col_name] == first_vector ) )

    all(equal_cols)
}

# raise exception if date, lat, or long are not the same across all dfs
stopifnot( check_same( df_list, "date") )
stopifnot( check_same( df_list, "lat") )
stopifnot( check_same( df_list, "long") )


merge_dfs =
    # OUTPUT: a single data frame with columns: date, lat, long, and the 7 variables
function(list_of_dfs)
{
    var_names = names(list_of_dfs)

    # begin with first data frame, then add columns
    df = list_of_dfs[[ var_names[1] ]]

    # remove first variable since already used
    var_names = var_names[-1]

    # from the list element v_name, extract the column v_name
    var_cols = lapply(var_names, function(v_name) list_of_dfs[[ v_name ]][ v_name ])
```

```
    # combine all columns into a single data frame
    do.call("cbind", c(df, var_cols))
}

df_all = merge_dfs(df_list)
dim(df_all)
```

```
## [1] 41472    10
```

Step 3

Reading in elevation data from "intlvtn.dat" and adding to our data frame created above. We'll use a similar approach to the one we used in step 1, making longitude values the column names and stacking the data, then adding in the latitudes. With this data however, there is some strange rounding (some lat/long values are rounded up and others are rounded down). We can see that they are very close, and so we'll use the lat/long values from step 1 to make sure all the data lines up correctly.

```
##########
# Step 3 #
##########

elevation_file = paste0(directory, "/intlvtn.dat")

# first line contains longitudes, which we'll use for columns
cols = readLines( elevation_file, n = 1 )
cols = strsplit( cols, " +")[[1]]

# read in remaining data, extract first column as latitudes
elevation_df = read.table( elevation_file, skip = 1)
latitude = elevation_df[, 1]
elevation_df = elevation_df[, -1]

# we have rounding differences, so we'll use lat/longs from the df in step 2
unique( df_all$lat ) - latitude
```

```
##  [1] -0.05  0.05 -0.05  0.05 -0.05  0.05 -0.05  0.05 -0.05  0.05 -0.05
## [12]  0.05 -0.05  0.05 -0.05  0.05 -0.05  0.05 -0.05  0.05 -0.05  0.05
## [23] -0.05  0.05
```

```
unique( df_all$long ) - as.numeric(cols)
```

```
##  [1] -0.05  0.05 -0.05  0.05 -0.05  0.05 -0.05  0.05 -0.05  0.05 -0.05
## [12]  0.05 -0.05  0.05 -0.05  0.05 -0.05  0.05 -0.05  0.05 -0.05  0.05
## [23] -0.05  0.05
```

```
cols = unique( df_all$long )
latitude = unique( df_all$lat )

# reshape data as done in step 1
names(elevation_df) = cols
elevation_df = stack(elevation_df)
elevation_df$lat = rep( latitude, length(col) )
```

```
names(elevation_df) = c("elevation", "long", "lat")

# making sure we can add elevation data with data from step 2
stopifnot( all( df_all$lat == elevation_df$lat ) )
stopifnot( all( df_all$long == elevation_df$long ) )

df_all$elevation = elevation_df$elevation

head(df_all, 3)
```

```
##         date  lat   long cloudhigh cloudlow cloudmid ozone pressure
## 1 1995-01-16 36.2 -113.8        26      7.5     34.5   304      835
## 2 1995-01-16 33.8 -113.8        20     11.5     32.5   304      940
## 3 1995-01-16 31.2 -113.8        16     16.5     26.0   298      960
##   surftemp temperature elevation
## 1    272.7       272.1   1526.25
## 2    279.5       282.2    612.94
## 3    284.7       285.2    328.62
```

Step 4

1. Plot temperature versus pressure. Color code the points by the value of cloudlow.

We first need to create a factor variable out of the continuous cloudlow variable. Cloudlow is right skewed, so we've decided to cut the variable using the quantiles, so each group has approximately equal size.

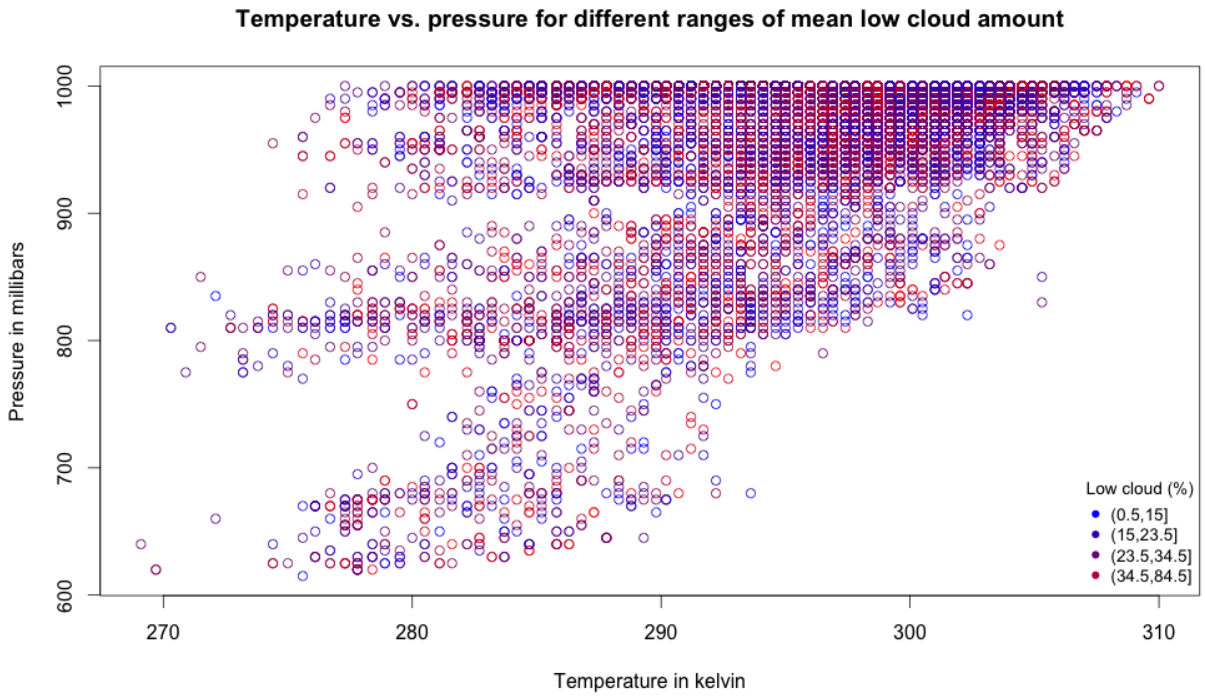
```
##########
# Step 4 #
##########

# find quantiles for creating factors with equally sized levels
cuts = quantile(df_all$cloudlow, na.rm = TRUE)
cloudlow_factor = cut( df_all$cloudlow, breaks = cuts )

# number of points in each level
table(cloudlow_factor)
```

```
## cloudlow_factor
##    (0.5,15]   (15,23.5] (23.5,34.5] (34.5,84.5]
##       10389       10841       10025       10103
```

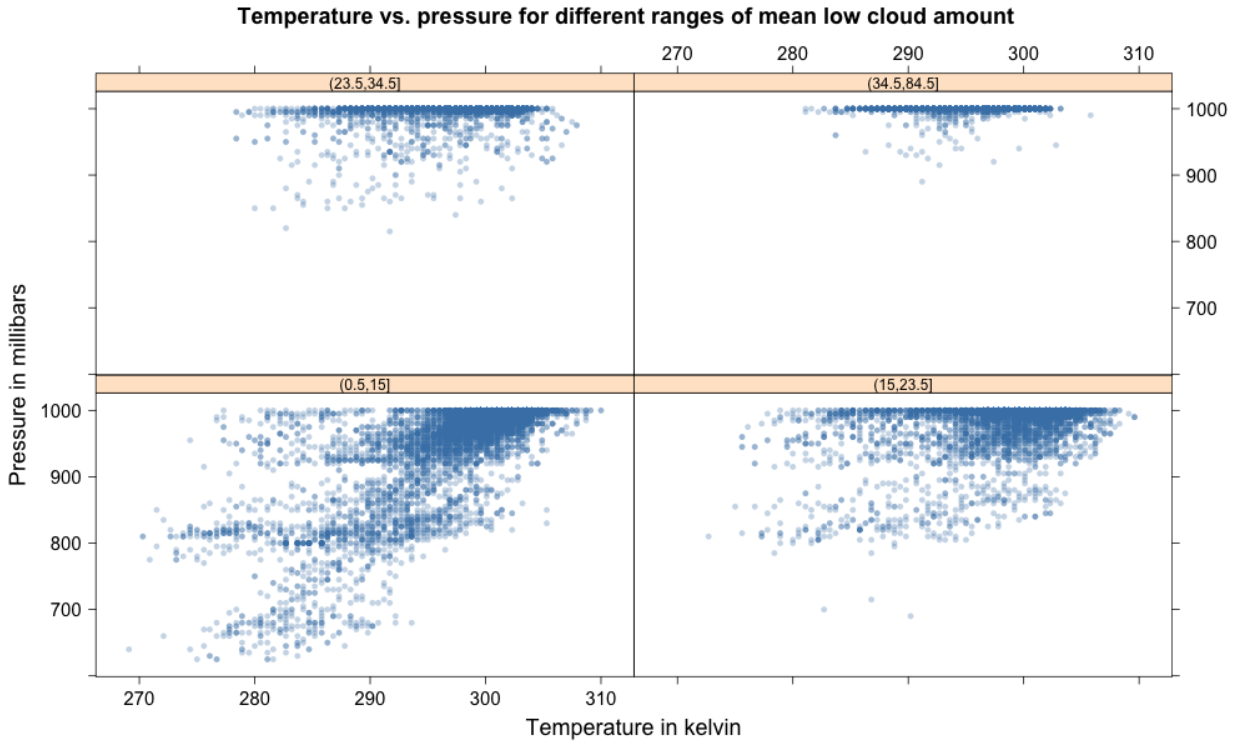
We then created a plot using different colors for the level of cloudlow. However, many of the points are overlapping (even while reducing the alpha level) and it's difficult to understand all the of the relationships. Therefore, we also created a xyplot showing the different levels of cloudlow in a separate plotting window. Recalling that each plotting window contains approximately the same number of points, it's now easy to see that for higher values of cloudlow, the pressure is close to 1000 regardless of temperature. For low values of cloudlow, the pressure values take on a much wider range, and we observe a somewhat positive linear between temperature and pressure.

```
# specs for saving plots as png
save_plot = function (file_name, width = 1000, height = 600) {
    png(file_name, width = width, height = height, pointsize = 16)
    file_name
```

```
}

plot_title = "Temperature vs. pressure for different ranges of mean low cloud amount"
plot_xlab = "Temperature in kelvin"
plot_ylab = "Pressure in millibars"

image_1 = save_plot("./images/step4_1a.png")

# all points on one plot
color_fun = colorRampPalette(c("blue", "red"))
plot(df_all$temperature, df_all$pressure,
     col = color_fun( length( unique(df_all$cloudlow) ) ),
     main = plot_title, xlab = plot_xlab, ylab = plot_ylab )

legend(x = "bottomright", legend = levels(cloudlow_factor),
       col = color_fun( length( unique( as.integer(cloudlow_factor) ) ) ),
       pch = 16, title = "Low cloud (%)", cex = .85, bty = "n")

invisible( dev.off() )

image_2 = save_plot("./images/step4_1b.png")

# panel plot, 2x2 by cloudlow
xyplot( pressure ~ temperature | cloudlow_factor, data = df_all, pch = 16,
        col = alpha("steelblue", 0.3),
        main = list( label = plot_title, cex = 1.5 ),
        xlab = list( label = plot_xlab, cex = 1.5 ),
        ylab = list( label = plot_ylab, cex = 1.5 ),
        scales = list( cex = 1.25 ))

invisible( dev.off() )

grid.raster( readPNG(image_1) )
```




```
grid.raster( readPNG(image_2) )
```



2. For points at the four corners of the spatial grid, display temperature over time.

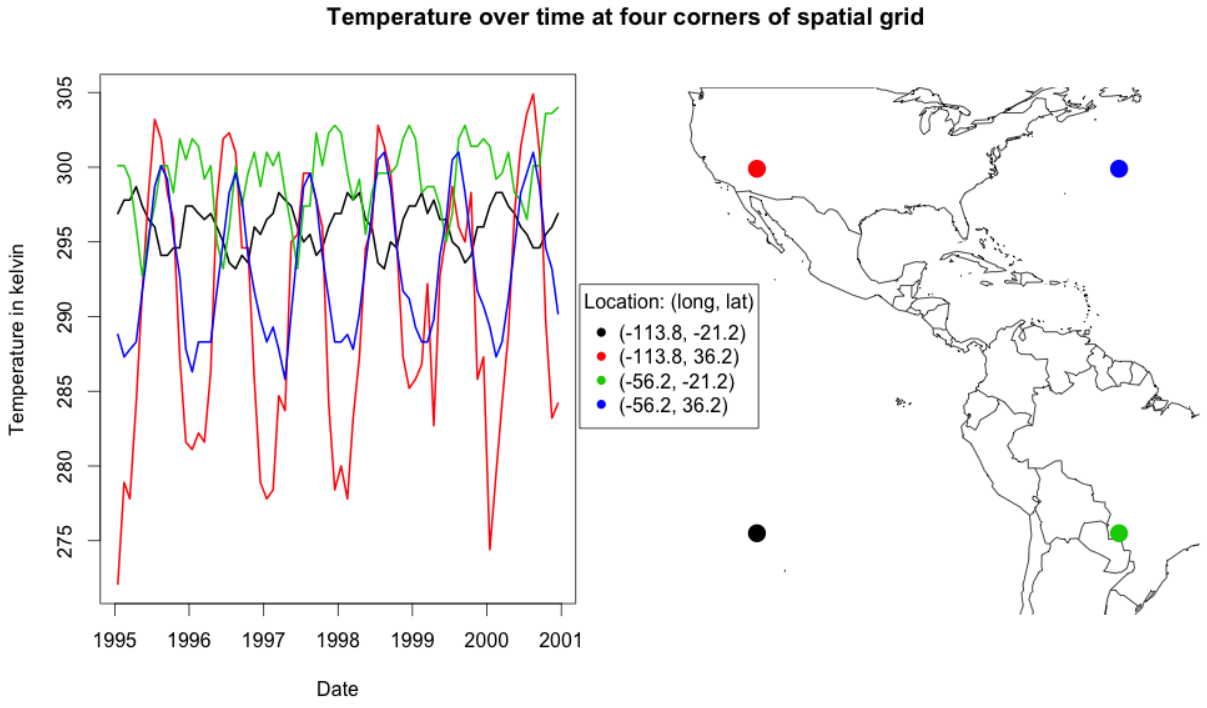
We'll first find the range of latitude and longitude values, and take combinations to find the four corners of the grid. We'll number the corners 1 through 4, and then merge with our original data to get only temperature readings at the four corners. Looking at the plot, we see a clear seasonal pattern to the temperatures. Comparing points in the northern and southern hemispheres, the peaks and valleys mostly align, especially in the northern hemisphere. We see the greatest temperature swings in-land in the US, followed by the Atlantic ocean. The Pacific ocean shows the least temperature variation while South America is warmest on average.

```
# find range of latitudes and longitudes
lat_range = range(df_all$lat)
long_range = range(df_all$long)

# df of all combinations of lat and long ranges (i.e.) locations of four corners of spatial grid
four_corners = setNames( expand.grid(lat_range, long_range), c("lat", "long") )
four_corners$corner = 1:4
four_corners
```

```
##     lat   long corner
## 1 -21.2 -113.8      1
## 2  36.2 -113.8      2
## 3 -21.2  -56.2      3
## 4  36.2  -56.2      4
```

```
keep_cols = c("lat", "long", "date", "temperature")
four_corner_temp = merge(four_corners, df_all[keep_cols])

# order by date within each corner
ordering = order( four_corner_temp$corner, four_corner_temp$date )
four_corner_temp = four_corner_temp[ordering, ]

by_corner = split(four_corner_temp, four_corner_temp$corner)

temp_range = range(four_corner_temp$temperature)

plot_world =
    # INPUT:
    #   - df: a data frame with lat and long columns
    #   - map_zoom: controls how much to zoom in on world map (larger number is less zoom)
function(df, map_zoom = 12)
{
    x_range = range( df$long ) + map_zoom * c(-1, 1)
    y_range = range( df$lat ) + map_zoom * c(-1, 1)

    map("world", xlim = x_range, ylim = y_range)
}

image_3 = save_plot("./images/step4_2.png")

#######################################
# left plot - temperatures over time #
#######################################

# setup plotting window
```

```r
par(mfrow = c(1, 2))
plot( by_corner[[1]]$date, by_corner[[1]]$temperature, type = "n",
      xlab = "Date", ylab = "Temperature in kelvin", ylim = temp_range,
      main = "")

# add lines to plot of different color
plot_temp = function(df) {
    points( df$date, df$temperature, type = "l", col = df$corner[1], lwd = 2)
}

invisible( lapply( by_corner, plot_temp ) )

##############################
# right plot - map / legend #
##############################

plot_world(four_corner_temp)
points(four_corners$long, four_corners$lat, pch = 16, cex = 2, col = four_corners$corner)

# text for legend, of the form (long, lat) for four corners
par(xpd = NA)
legend_text = paste0("(", four_corners$long, ", ", four_corners$lat, ")")
legend(x = -142, y = 18, legend = legend_text, pch = 16, col = four_corners$corner,
       title = "Location: (long, lat)")
title( main = "Temperature over time at four corners of spatial grid", outer = TRUE, line = -2 )

invisible( dev.off() )

grid.raster( readPNG(image_3) )
```



3. For all points on the grid, compute the average and standard deviation for each of the 7 variables across time.

We'll group the data by latitude and longitude, and compute either the mean or the standard deviation.

```
agg_mean_sd =
    # INPUT:
    #   - df: the data frame to aggregate over
    #   - col_names: the columns to keep for computing the summary_fun
    #   - summary_fun: the function to pass to aggregate, we'll use mean / sd
    #
    # DOC: for df subset by col_names, group by lat and long and compute summary_fun
function( df, col_names, summary_fun )
{
    aggregate( df[ col_names ], list( lat = df$lat, long = df$long), summary_fun )
}

# get the 7 variable names, (elevation added for problem 5)
var_names = c(names(df_list), "elevation")

mean_df = agg_mean_sd( df_all, var_names, mean )
sd_df = agg_mean_sd( df_all, var_names, sd)

# the number of results we expect
length( unique( df_all$lat ) ) * length( unique( df_all$long ) )
```

```
## [1] 576
```

```
# number of results we got
dim(mean_df); dim(sd_df)
```

```
## [1] 576  10
```

```
## [1] 576  10
```

```
head(mean_df, 3)
```

```
##     lat   long cloudhigh cloudlow cloudmid    ozone pressure surftemp
## 1 -21.2 -113.8  1.993056 37.17361 5.777778 268.2500     1000 296.2417
## 2 -18.8 -113.8  1.041667 39.36111 4.055556 265.7500     1000 296.3292
## 3 -16.2 -113.8  0.687500 40.22222 3.819444 262.7778     1000 296.7403
##   temperature elevation
## 1    296.1083         0
## 2    296.3069         0
## 3    296.7736         0
```

```
head(sd_df, 3)
```

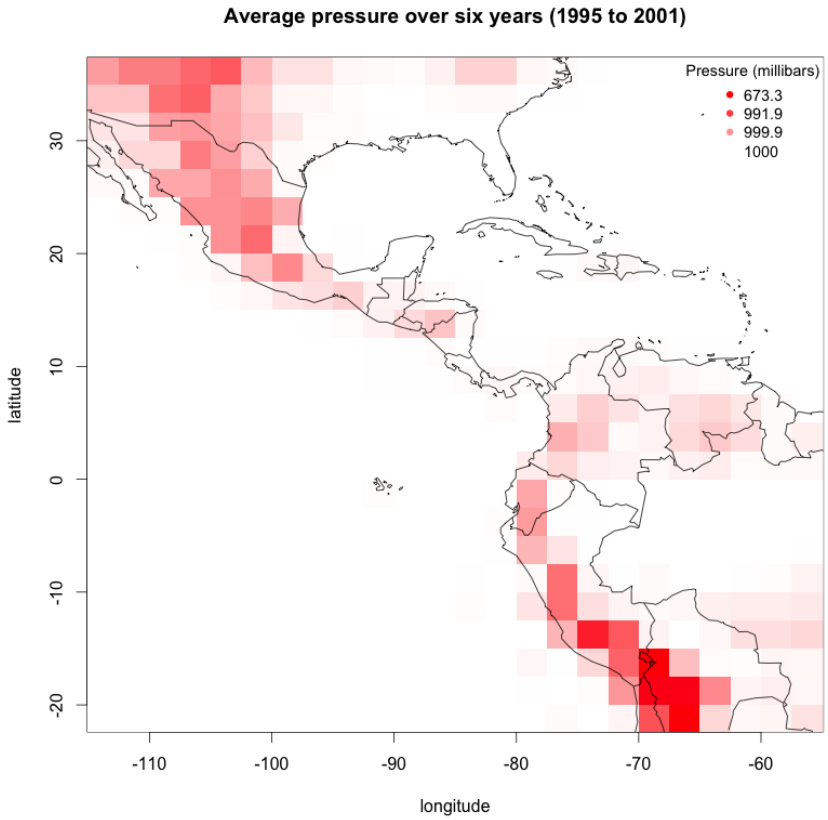
```
##     lat   long cloudhigh cloudlow cloudmid     ozone pressure surftemp
## 1 -21.2 -113.8  2.769925 5.782538 3.817942 12.259805        0 1.499178
## 2 -18.8 -113.8  1.915007 5.896539 3.185084 10.762153        0 1.251077
```

```
## 3 -16.2 -113.8  1.046246 6.741656 3.144754  9.245055        0 1.163610
##   temperature elevation
## 1    1.475600         0
## 2    1.358264         0
## 3    1.346356         0
```

4. Display the average value for pressure computed in the previous question on a map.

We first need to cut the continuous pressure variable to create a categorical variable. Looking at the values of pressure, we see a majority of the values are 1000 or very close to 1000. We'll use the quantile function to create 4 groups which are as close to equally sized as possible. It's clear from the plot, that higher pressures near 1000 occur over the ocean and the coasts, while the lower pressures occur in-land. Strangely, this high pressure also occurs over the Brazilian rain forest.

```
image_4 = save_plot("./images/step4_4.png", 800, 800)

# get average pressure at each location
pressure_mean = tapply( mean_df$pressure, list(mean_df$long, mean_df$lat), mean )
x = as.numeric( rownames( pressure_mean ) )
y = as.numeric( colnames( pressure_mean ) )

# plot heat mpa
color_fun = colorRampPalette( c("red", "white") )
image(x, y, pressure_mean, xlab = "longitude", ylab = "latitude",
      col = color_fun( max(mean_df$pressure) ),
      main = 'Average pressure over six years (1995 to 2001)')

# add the world map
map( "world", xlim = range(x), ylim = range(y), add = TRUE )

# add the legend
cuts = round( quantile(mean_df$pressure, probs = seq(0, 1, 0.2))[1:4], 1 )
legend('topright', legend = cuts,
       pch = 16, col = color_fun( length(cuts) ),
       title = "Pressure (millibars)", cex = 0.9, bty = "n" )

invisible( dev.off() )
grid.raster( readPNG(image_4) )
```



5. Display average surface temperature versus elevation.

In order to display surface temperature versus elevation, we went back to problem 3 and added elevation as a variable which is averaged over time. Since elevation doesn't change over time, we'll get the standard deviation of elevation is zero, and the mean elevation is simply the elevation at each location on the grid. We see from the plot that at sea level (elevation = 0) the average surface temperature ranges from about 290 to 302 degrees kelvin. As elevation increases, we see that average surface temperature decreases.

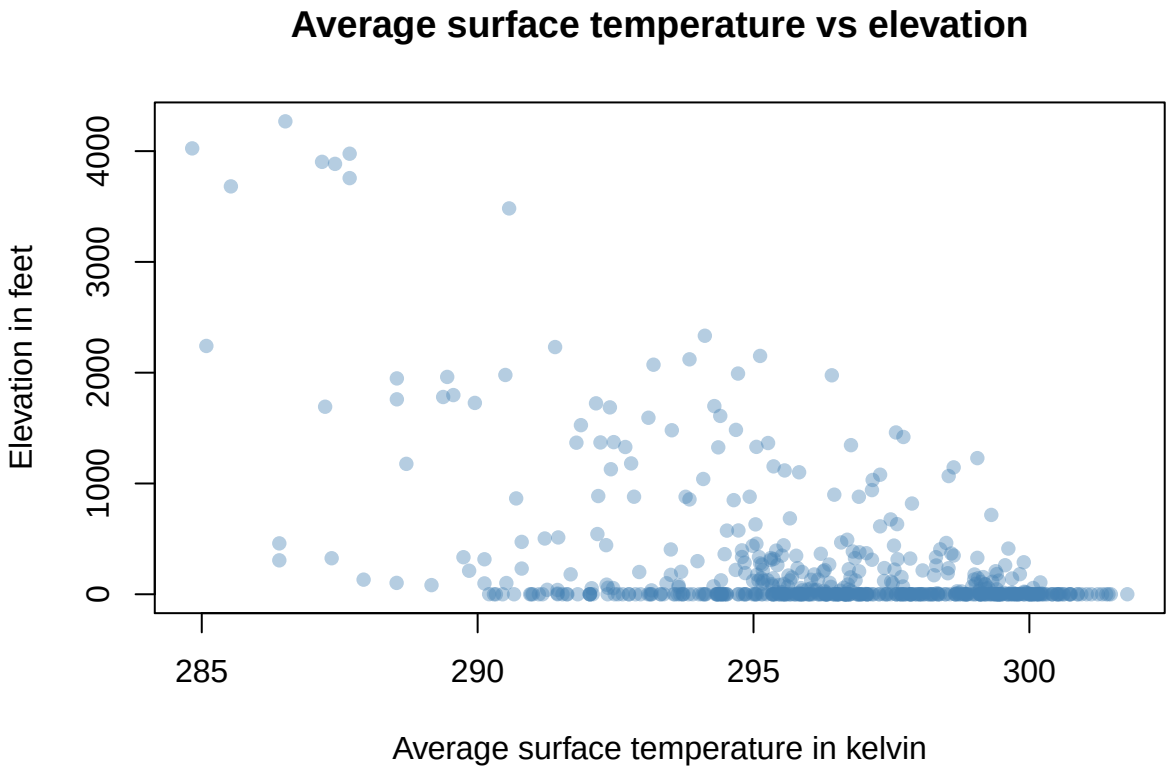
```
# showing mean(elevation) as computed in problem 3 equals elevation
summary( mean_df$elevation )
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0     0.0   257.8   181.5  4269.0
```

```
all( sd_df$elevation == 0 )
```

```
## [1] TRUE
```

```
plot( mean_df$surftemp, mean_df$elevation, pch = 16,
      col = alpha("steelblue", 0.4),
      main = "Average surface temperature vs elevation",
      ylab = "Elevation in feet",
      xlab = "Average surface temperature in kelvin")
```


Average surface temperature vs elevation
Elevation in feet
0
1000
2000
3000
4000
285
290
295
300
Average surface temperature in kelvin