Post 2

Jackie Zou

12/2/2017

Introduction to Heat Maps in R

What are heat maps? Heat maps are a type of data visualization tool that match colors on a gradient to scaled numberical values; that is, they replace numerical values in a matrix with a color for fast access to possible trends or patterns in the data. They're extremely useful for giving your reader and yourself with a visual summary of what's going on, and they have applications in lots of different types of data sets. In this post, I'll guide you through 3 of the different ways you can make heat maps in R: one with a built-in R function, one with the package gplots, and one with plotly (which has the bonus of also being interactive).

Let's begin!

1) Making Heat Maps with Built-in R Function

```
#Loading dplyr for some data frame manipulation we'll be doing throughout this post!
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

We'll start learning how to do a simple heat map with the built-in R function heatmap(). First, we're going to want to load in some data. For this first example, let's look at some cheery US crime data organized by state.

```
#Loading the crime data
crime_url <- "https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/
datasets/USArrests.csv"
crime_vars <- c("State", "Murder", "Assault", "UrbanPop", "Rape")
crime <- read.csv(file = crime_url,
                  col.names = crime_vars) %>% select(1:5)

head(crime)
```

```
##        State Murder Assault UrbanPop Rape
## 1    Alabama   13.2     236       58 21.2
## 2     Alaska   10.0     263       48 44.5
## 3    Arizona    8.1     294       80 31.0
## 4   Arkansas    8.8     190       50 19.5
## 5 California    9.0     276       91 40.6
## 6   Colorado    7.9     204       78 38.7
```

Right now, the row names are just 1, 2, 3 , etc, but we want to make it into something that makes more sense, like the corresponding state names we have already.

```
#Changing our row names to the names of each state
row.names(crime) <- crime$State
head(crime)
```

```
##                 State Murder Assault UrbanPop Rape
## Alabama       Alabama   13.2     236       58 21.2
## Alaska         Alaska   10.0     263       48 44.5
## Arizona       Arizona    8.1     294       80 31.0
## Arkansas     Arkansas    8.8     190       50 19.5
## California California    9.0     276       91 40.6
## Colorado     Colorado    7.9     204       78 38.7
```

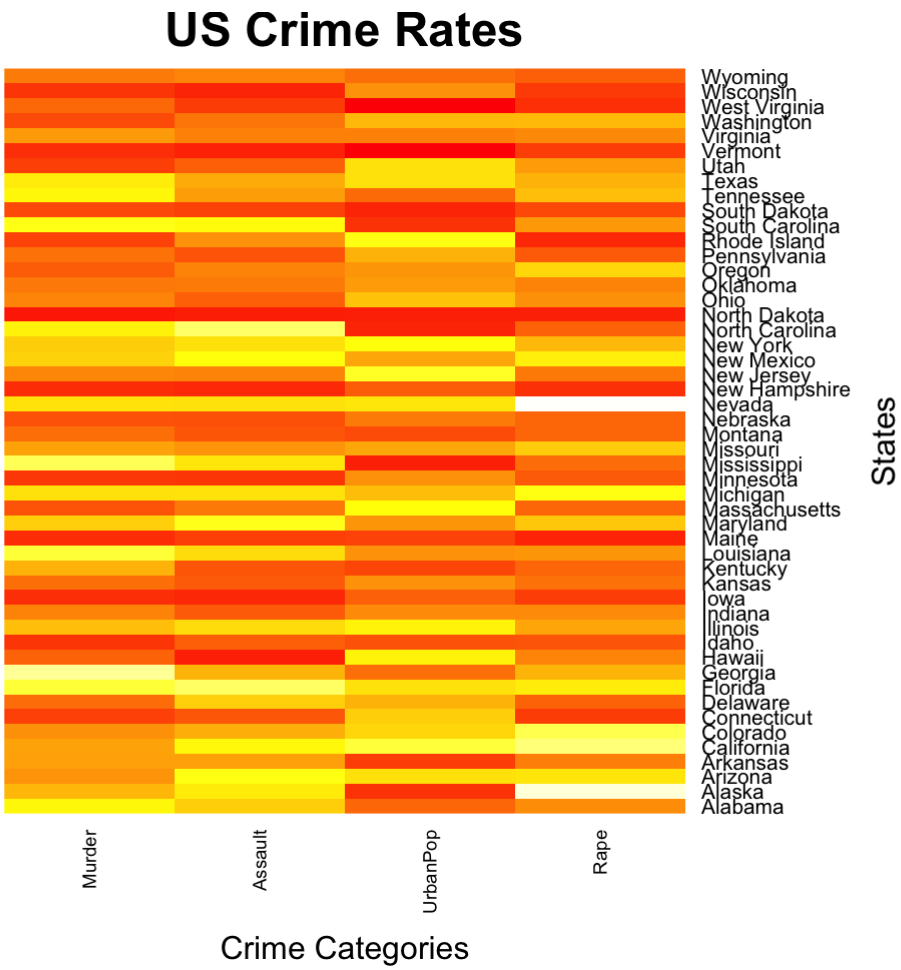
Now that that's done, we're going to drop the now-redundant State column and then turn our data frame into something that our heatmap() function can take in, which is a matrix.

```
#Converting our csv into a matrix that can be passed into out heatmap function, excludin
g now the column of names
crime_withoutnames <- select(crime, 2:5)
crime_matrix <- data.matrix(crime_withoutnames)
head(crime_matrix)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

Now that our matrix is prepared, all we need to do is plug it into our heatmap function!

```
#Making our basic heatmap
heatmap(x = crime_matrix,
        Rowv=NA, Colv=NA,            # turns off dendrograms
        col = heat.colors(300),      # designates color scheme and steps
        scale="column",              # how to scale colors by (row or column)
        main = "US Crime Rates",     # title for heat map
        xlab = "Crime Categories",   # X Label
        ylab = "States",             # Y Label
        margins = c(5, 7),           # margin around x and y labels
        cexCol = .7)                 # scales label text
```



Beautiful! Let's do deeper into some basic inputs into the base R heatmap function:

- "x" is a matrix of values that you wish to turn into a heatmap, with the row names appearing on the side and plotted against out different columns.
- "Rowv" and "Colv" have to do with the drawing of row and column dendrograms, which for our purposes we will be supressing through the passing of an "NA" argument.
- "col" determines what set of colors are going to be used to display the data; built in options include heat.colors, terrain.colors, topo.colors, and cm.colors. The number in the parenthesis correspond to how many colors in the set; the more colors there are the more gradual our gradient. We'll explore this aspect of color a little more later.
- "scale" takes in a character string that determines how the colors in cells should be assigned in relation to one another; in this case, because we want to compare values within the same category, we passed in "column." The default is by row, which would be helpful in circumstances where, for example, each of the

columns referenced measureing the same aspect of one thing.

2) Heat Maps in gplots

Our second way of creating heat maps in R involves loading two packages: gplots and RColorBrewer, the code for which I've included below. For my own purposes of running this I've commented out the installation command, but you should run it in your console if it's your first time using it.

```
#install.packages("gplots")
library("gplots")
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
#install.packages("RColorBrewer")
library("RColorBrewer")
```

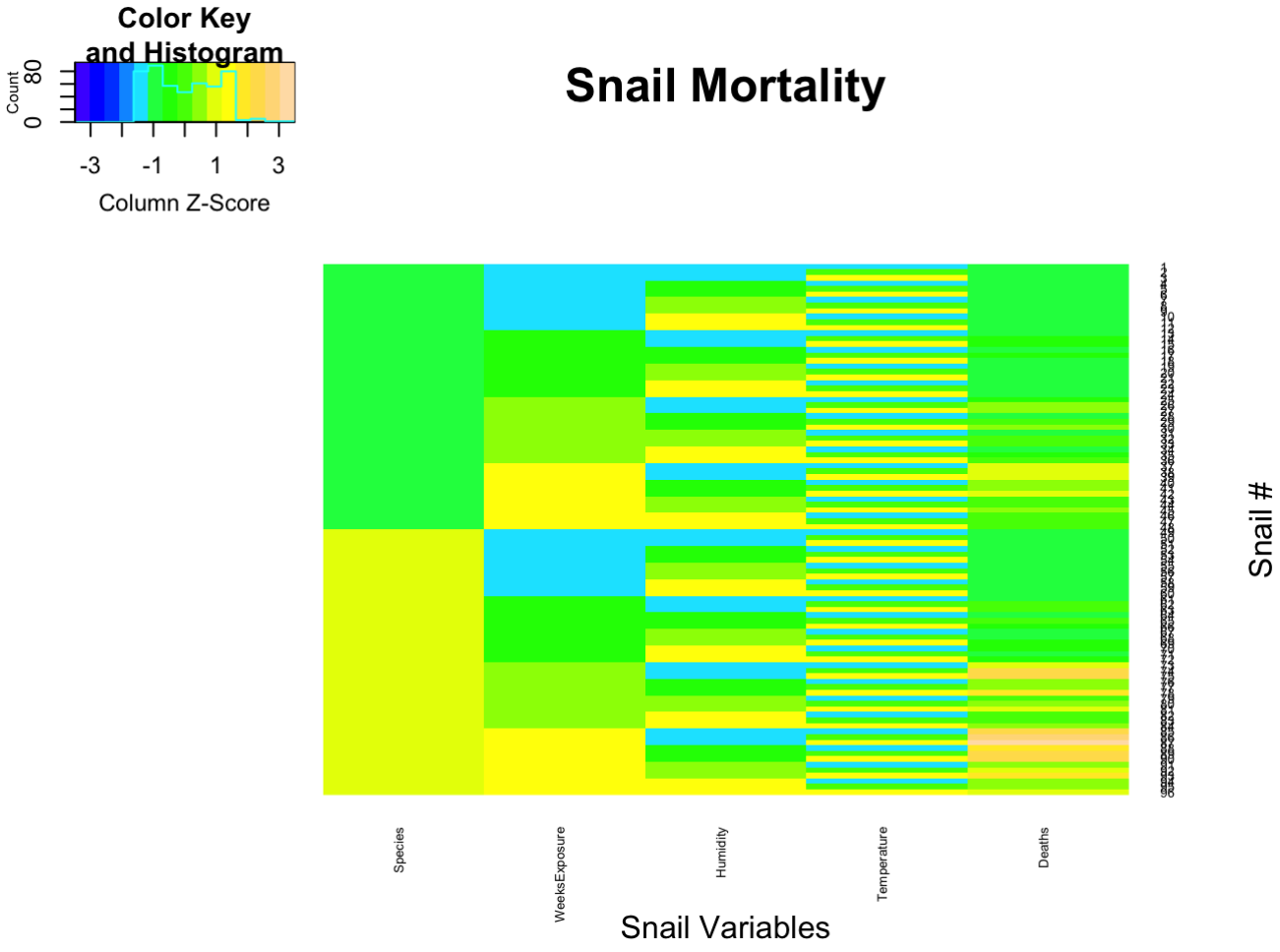
For this example, let's use a fun data set we found online called "Snail Mortality" and prepare it in the same way we did for US Crime, loading it and then converting it into a data matrix.

```
#Loading the snail data and removing irrelevant rows
snail_url <- "https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/
MASS/snails.csv"
snail_vars <- c("Snail No.", "Species", "WeeksExposure", "Humidity", "Temperature", "Dea
ths", "Number")

snails <- read.csv(file = snail_url, col.names = snail_vars, sep = ",")

#Converting to matrix and assigning snail number to row names
snails_matrix <- data.matrix(select(snails, 2:(ncol(snails)-1)))
rownames(snails_matrix) <- snails$Snail.No.

head(snails_matrix)
```

```
##   Species WeeksExposure Humidity Temperature Deaths
## 1       1             1     60.0          10      0
## 2       1             1     60.0          15      0
## 3       1             1     60.0          20      0
## 4       1             1     65.8          10      0
## 5       1             1     65.8          15      0
## 6       1             1     65.8          20      0
```

Now let's pass in our data matrix with the function heatmap.2()! There are a lot of parameters you can pass into this function, but here are the basics that are important to consider:

```
heatmap.2(snails_matrix,
  main = "Snail Mortality",         # heat map title
  density.info="histogram",         # adds a density plot to color key
  trace="none",                     # turns off trace lines inside the heat map
  xlab = "Snail Variables",         # string label for X axis
  ylab = "Snail #",                 # string label for Y axis
  margins =c(5,5),                  # defines margins between variable names and labels
  col=topo.colors,                  # defines color scheme
  Rowv = NULL,                      # removes dendrogram options
  Colv = NULL,                      # removes dendrogram options
  dendrogram = "none",              # removes dendrogram options
  scale = "column",                 # how to scale colors by (row or column)
  cexCol = .5,                      # designates font scale of column names
  cexRow = .5)                      # designates font scale of row names
```



Custom Colors in RColorBrewer

Note that a great advantage of this package over base R is that we have a lot more options in terms of customization, including a color key that can display density data in the top left. What we can also do is decide for ourselves what colors we want to use in our heatmap; in the previous two examples we've only used default palletes provided in R, but let's try some new *pretty* things with the RColorBrewer package.

The syntax is fairly uncomplicated; the function takes in a vector of color names, which you can access a full list of in R at http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf (http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf). The second part of the argument specifies the gradient

steps that you want your pallete to have; for those unfamiliar with deisgn gradients they're essentially the number of in-between, transition colors in the gradient we created. A smaller number will mean less gradual transitions and a larger number will mean smoother transitions.

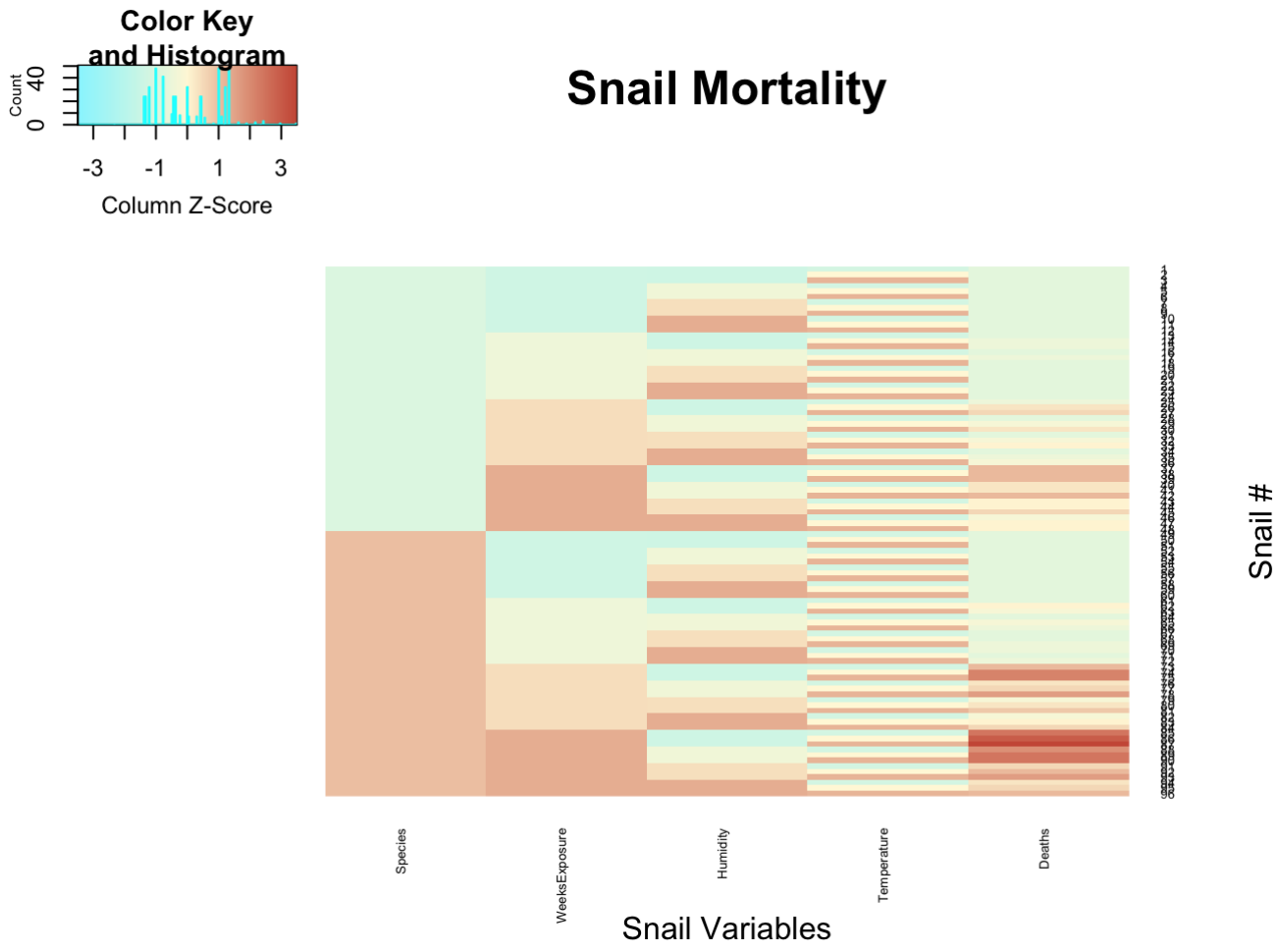
Note also that the gradient will be created with colors in the order that they're passed in, so for readability's sake you're probably going to want to list your colors from least to most intense.

First, we'll make our pallete and assign it to a variable.

```
#defining our pallete
pallette1 <- colorRampPalette(c("cadetblue1", "cornsilk", "coral3"))(299)
```

Now we'll take the pallete we just made and pass it into the "col =" argument of our heatmap.2 function, and viola! Your custom heat map.

```
heatmap.2(snails_matrix,
  main = "Snail Mortality",
  density.info="histogram",
  trace="none",
  xlab = "Snail Variables",
  ylab = "Snail #",
  margins =c(5,5),
  col=pallette1,                    # defines color scheme as our custom pallete
  Rowv = NULL,
  Colv = NULL,
  dendrogram = "none",
  scale = "column",
  cexCol = .5,
  cexRow = .5)
```



3) Heat Maps using plotly

A third way to make heat maps that we'll be going over in this tutorial is through a great package called plotly. Plotly creates beautiful plots that allow you to hover over and interact with the graphs that you make. I won't go too much into detail on the advantages/uses of interactive plots here (I covered this more in depth in my first post, which I recommend if you're interested in learning more about other capabilities of this package!), but they're extremely powerful for displaying your data in an engaging and efficient way.

First, we'll install/load the package plotly. Note that it will also likely install the dependency ggplot2 if you don't already have it. We'll also load stringr, because for this example I decided to do some simple string manipulation to make our data more readable.

```
#install.packages("plotly")
library("plotly")
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library("stringr")
```

First, let's load the data, which has to do with measuring air quality in NYC on specific days of the year.

```
#Loading the nyc data
nyc_url <- "https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/da
tasets/airquality.csv"
nyc <- read.csv(file = nyc_url)
head(nyc)
```

```
##   X Ozone Solar.R Wind Temp Month Day
## 1 1    41     190  7.4   67     5   1
## 2 2    36     118  8.0   72     5   2
## 3 3    12     149 12.6   74     5   3
## 4 4    18     313 11.5   62     5   4
## 5 5    NA      NA 14.3   56     5   5
## 6 6    28      NA 14.9   66     5   6
```

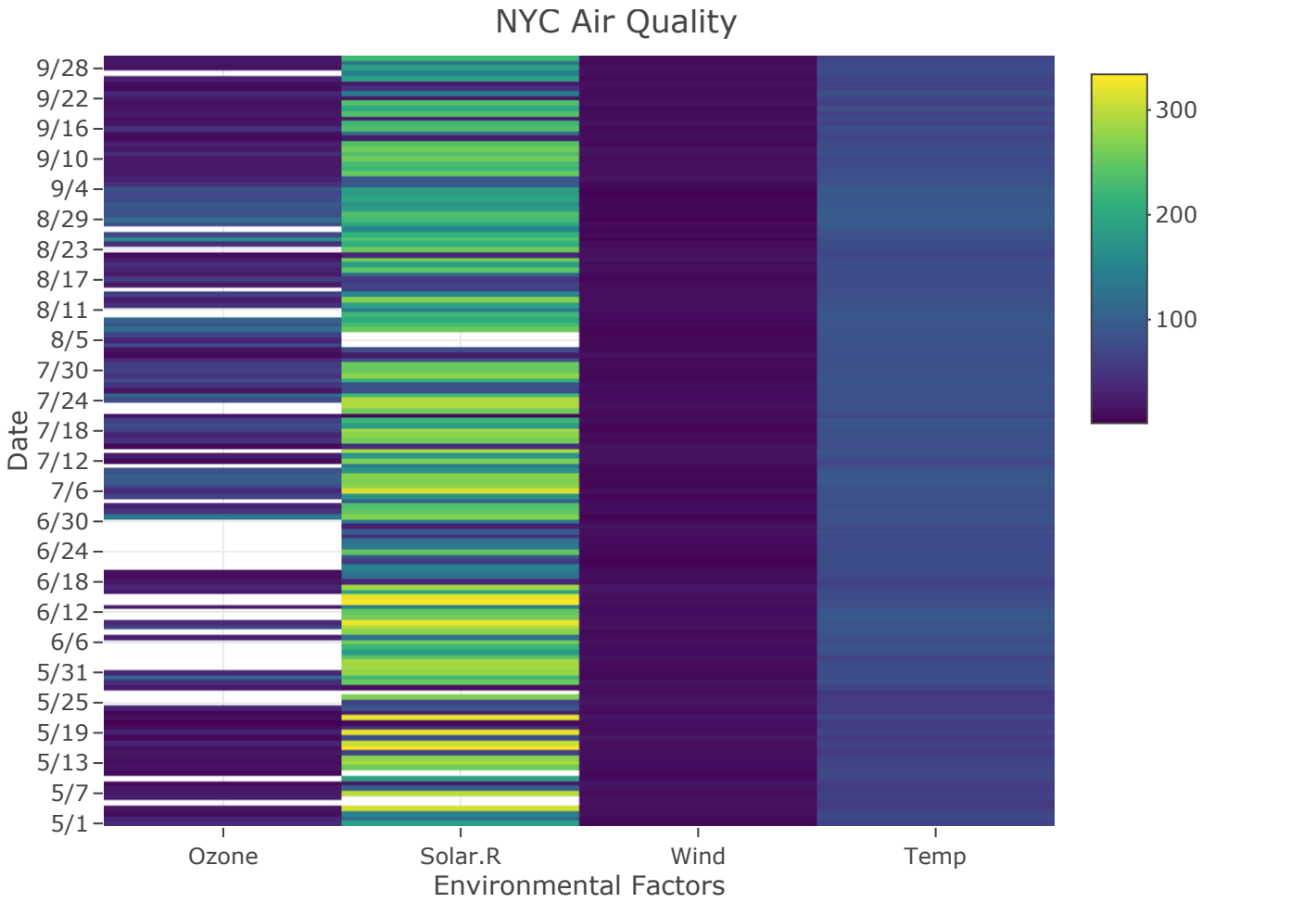
Next, like we did for the others, we're going to convert our data frame into a matrix, with the extra step that we're going to use our Month and Day columns to contruct dates that we can use for our row names.

```
#Creating our data matrix with relevant columns
nyc_matrix <- data.matrix(select(nyc, 2:5))

#Assigning dates from original Month and Day columns
dates<- paste(nyc$Month, nyc$Day, sep = "/")
rownames(nyc_matrix) <- dates

head(nyc_matrix)
```

```
##     Ozone Solar.R Wind Temp
## 5/1    41     190  7.4   67
## 5/2    36     118  8.0   72
## 5/3    12     149 12.6   74
## 5/4    18     313 11.5   62
## 5/5    NA      NA 14.3   56
## 5/6    28      NA 14.9   66
```

Now to make the actual plot! They syntax of plot_ly() differs a little bit from the other two; here we specify the x as the columns, y as the rows, and the z as the data set we'll be scaling over.

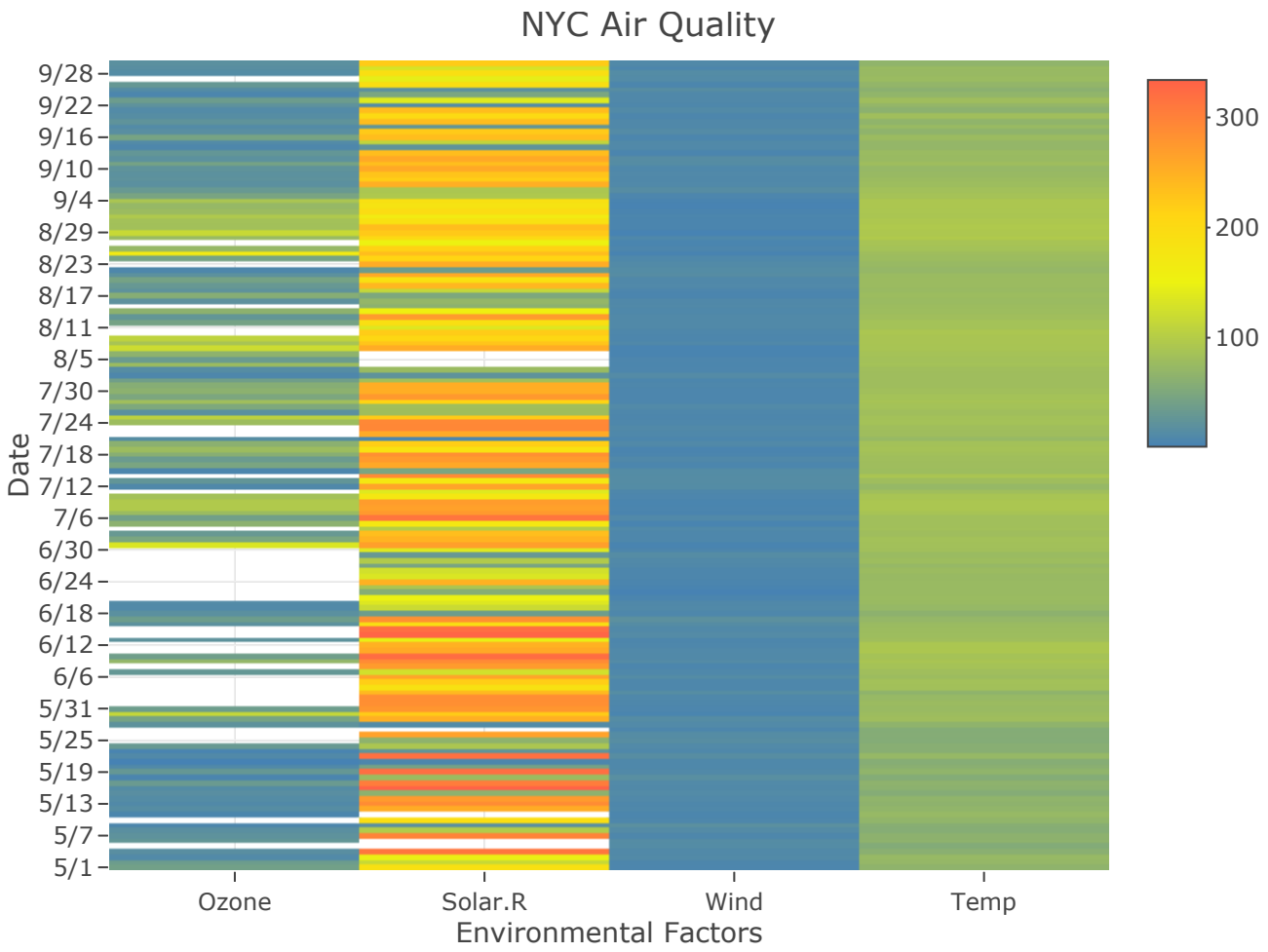
```
plot_ly(x=colnames(nyc_matrix),                                    # designating x variabl
es
        y=rownames(nyc_matrix),                                    # designating y variabl
es
        z = nyc_matrix,                                            # assigning our matrix
        type = "heatmap"                                           # designating the type
 of plot
        ) %>%
  layout(title = "NYC Air Quality",                                # main title
         xaxis = list(title = "Environmental Factors"),            # x-axis title (must be
 in a list)
         yaxis = list(title ="Date"))                              #y-axis title (must be
 in a list)
```



Note also that in our data set we had NA values, which here show up as blank spaces on our heatmap.

We can also, like the other two, pass in our own colors using colorRamp(), which also takes in a vector of colors to construct the gradient with.

```
plot_ly(x=colnames(nyc_matrix),
        y=rownames(nyc_matrix),
        z = nyc_matrix,
        type = "heatmap",
        colors = colorRamp(c("steelblue", "yellow1", "tomato"))      #optional custom co
lor argument
        ) %>%
  layout(title = "NYC Air Quality",
         xaxis = list(title = "Environmental Factors"),
         yaxis = list(title ="Date"))
```



Now go forth and have fun!

Play around with heat maps on different data sets! They're extremely easy to use and fast ways to find possible correlations in random data sets, and are beautiful ways to add a little *spice* to your reports. If you're looking for more on the possibilities of heat plots, like those that are superimposed on geographical maps, those with dendrograms, etc, there's a lot more to learn on the topic right here (https://plot.ly/r/heatmaps/ (https://plot.ly/r/heatmaps/)) and in some of the references I'll list below.

Thanks for reading, may your future plots be full of engaging and informative colors!

References:

- http://www.r-graph-gallery.com/215-interactive-heatmap-with-plotly/ (http://www.r-graph-gallery.com/215-interactive-heatmap-with-plotly/)

- https://stackoverflow.com/questions/28546637/r-changing-the-size-of-a-heatmap-2-rowname-column (https://stackoverflow.com/questions/28546637/r-changing-the-size-of-a-heatmap-2-rowname-column)
- http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf (http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf)
- https://plot.ly/r/heatmaps/ (https://plot.ly/r/heatmaps/)
- http://sebastianraschka.com/Articles/heatmaps_in_r.html (http://sebastianraschka.com/Articles/heatmaps_in_r.html)
- https://plot.ly/r/figure-labels/#figure-labels-for-2d-charts (https://plot.ly/r/figure-labels/#figure-labels-for-2d-charts)
- http://flowingdata.com/2010/01/21/how-to-make-a-heatmap-a-quick-and-easy-solution/ (http://flowingdata.com/2010/01/21/how-to-make-a-heatmap-a-quick-and-easy-solution/)
- https://vincentarelbundock.github.io/Rdatasets/datasets.html (https://vincentarelbundock.github.io/Rdatasets/datasets.html) (for a huge collection of data sets to use)