A Minimal Bookdown Book

James Simkins

2020-05-19

Contents

Preface

This is the very first part of the book.

Chapter 1

Introduction

This is the first real chapter.

Chapter 2

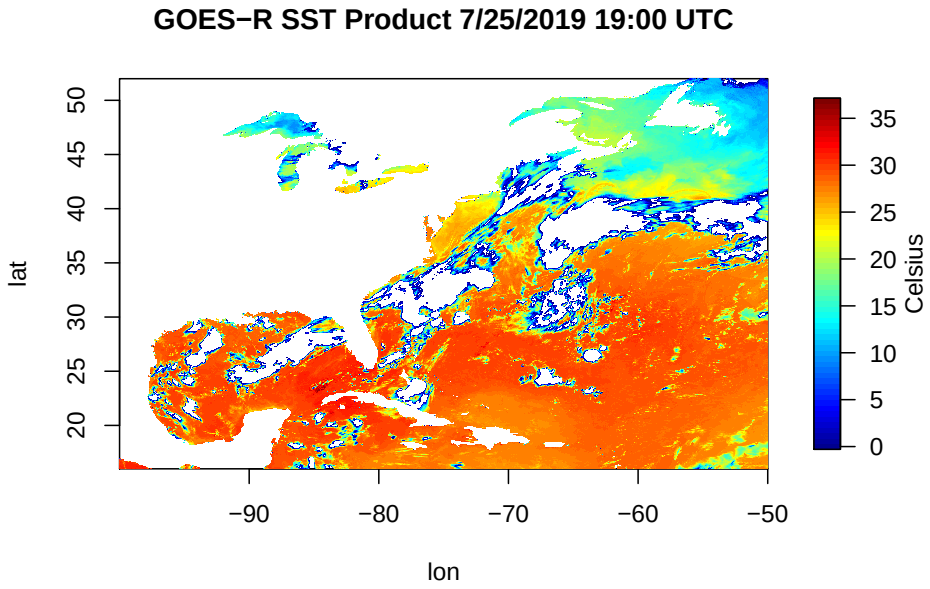
Diving In

2.1 How does an R script generally flow?

Now it's time to get your hands dirty. We can sit and chat about all these things R can do and how to do them, but you won't retain that information until you use R yourself. This will be confusing at first if you're new to this type of programming, but the longer you spend with it, the more sense it will make.

A typical R script will generally look like this:

```
# James Simkins
# Load libraries
library(ncdf4)
library(fields)
ncFile <- ncdf4::nc_open("~/Documents/Github/geog473-673/datasets/OR_ABI-L2-SSTF-M3_G16_s20192081
sstK <- ncdf4::ncvar_get(nc=ncFile, varid="SST")
lat <- ncdf4::ncvar_get(nc=ncFile, varid="latitude")
lon <- ncdf4::ncvar_get(nc=ncFile, varid="longitude")
# convert sst from Kelvin to Celsius
sstC <- sstK - 273.15
# remove values below 0C
sstC[sstC < 0] = NA
# Plot the matrix
fields::image.plot(x=lon, y=lat, z=sstC, legend.lab="Celsius")
title("GOES-R SST Product 7/25/2019 19:00 UTC")
```



2.2 Now Let's Break Down This Script

```
# Load libraries
library(ncdf4)
library(fields)
```

In R, we need to call on packages/libraries that we want to load in. We do this via the "library()" function or the "require()" function - both do the same thing. Notice that we do this at the beginning of the script because R reads line by line and we need these loaded before we can use the functions within the packages.

```
# Load libraries
library(ncdf4)
library(fields)
########################################

ncFile <- ncdf4::nc_open(filename="~/Documents/Github/geog473-673/datasets/OR_ABI-L2-SS
sstK <- ncdf4::ncvar_get(nc=ncFile, varid="SST")
lat <- ncdf4::ncvar_get(nc=ncFile, varid="latitude")
lon <- ncdf4::ncvar_get(nc=ncFile, varid="longitude")
```

Object = something. In this case, object name is ncFile and it holds an opened NetCDF file. We open this file via the "nc_open" function that's within the "ncdf4" library. Note the "ncdf4::" syntax. This is NOT necessary for coding in R. Once you load in the library, R knows what you mean when you type in a

function such as "nc_open". I added it here so you know where these functions are coming from.

Notice that I use "<-" for objects and "=" for arguments within the function. This is key, as I can use either "<-" or "=" for objects (like ncFile or sstK) but I MUST use "=" within the function 'walls' (the parantheses).

For those of you not used to NetCDF files, they're an efficient filetype heavily used in physical sciences. Within each file, metadata (time, latitude info, longitude info, projection, etc.), and variables (sea surface temperature, latitude points, longitude points, chlorophyll, etc.) are stored in these. We open the netcdf file and then extract what we want out of it using "ncvar_get", which is short for "netcdf variable get". Confused about how to use ncvar_get? Try:

```
# Load libraries
library(ncdf4)
library(fields)
ncFile <- ncdf4::nc_open(filename="~/Documents/Github/geog473-673/datasets/OR_ABI-L2-SSTF-M3_G16_
sstK <- ncdf4::ncvar_get(nc=ncFile, varid="SST")
########################################

help(ncvar_get)
```

This is a really attractive feature within R and exists for every function within an official R library. Now, back to the script...

```
# Load libraries
library(ncdf4)
library(fields)
ncFile <- ncdf4::nc_open(filename="~/Documents/Github/geog473-673/datasets/OR_ABI-L2-SSTF-M3_G16_
sstK <- ncdf4::ncvar_get(nc=ncFile, varid="SST")
########################################
# convert sst from Kelvin to Celsius
sstC <- sstK - 273.15
```

The SST variable from the netCDF file was in Kelvin and we want to convert it to Celsius. Right now, sstK is a matrix. How do I know this? Look at your environment, or simply type into your console:

```
# Load libraries
library(ncdf4)
library(fields)
ncFile <- ncdf4::nc_open(filename="~/Documents/Github/geog473-673/datasets/OR_ABI-L2-SSTF-M3_G16_
sstK <- ncdf4::ncvar_get(nc=ncFile, varid="SST")
########################################

class(sstK)
```

```
## [1] "matrix"
```

Class is a useful function that is loaded with the base library everytime you fire up R. It tells us what type of object we have. Now that we know this is a matrix, we can subtract 0 Celsius, or 273.15 Kelvin. When we have a matrix in R and perform any math on it, it does that math on each and every matrix value.

#Quick and Dirty Quality Control

There are bad values that crept into the dataset and we need to convert all of them to NaN (aka Not A Number…also known as NA (Not Available) in R). How do we know there are bad values in this dataset?

```
# Load libraries
library(ncdf4)
library(fields)
ncFile <- ncdf4::nc_open(filename="~/Documents/Github/geog473-673/datasets/OR_ABI-L2-SS
sstK <- ncdf4::ncvar_get(nc=ncFile, varid="SST")
#######################################

summary(as.vector(sstK))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -999.0   286.9   300.0   278.2   302.0   310.0 2175822
```

summary() is another great base function. In order to use it on a matrix, we need to convert it to a vector - summary can't do 2 dimensional objects like matrices, it needs a one dimensional vector of numbers. The "as.vector" function just says hey R read in sstK as a vector for me would ya? Notice the 'Min' in the summary output. -999? No way is that a valid Kelvin value, especially since we subtract another 273.15 to this number to make the the Celsius matrix. So clearly we have some bad data that we need to convert to NA's. We do this by…

```
library(ncdf4)
library(fields)
ncFile <- ncdf4::nc_open("~/Documents/Github/geog473-673/datasets/OR_ABI-L2-SSTF-M3_G1
sstK <- ncdf4::ncvar_get(nc=ncFile, varid="SST")
lat <- ncdf4::ncvar_get(nc=ncFile, varid="latitude")
lon <- ncdf4::ncvar_get(nc=ncFile, varid="longitude")
# convert sst from Kelvin to Celsius
sstC <- sstK - 273.15
# remove values below OC
#######################################

sstC[sstC < 0] = NA
```

This line reads as: sstC where sstC is less than 0 equals NA. The brackets here can be thought of as the 'condition', that is what you're looking to change. This is called a vector operation, which we will get more into later but these are

important becuase it's far faster to do this than a for loop.

```
library(ncdf4)
library(fields)
ncFile <- ncdf4::nc_open("~/Documents/Github/geog473-673/datasets/OR_ABI-L2-SSTF-M3_G16_s20192081
sstK <- ncdf4::ncvar_get(nc=ncFile, varid="SST")
lat <- ncdf4::ncvar_get(nc=ncFile, varid="latitude")
lon <- ncdf4::ncvar_get(nc=ncFile, varid="longitude")
# convert sst from Kelvin to Celsius
sstC <- sstK - 273.15
# remove values below OC
########################################

sstC[sstC < 0] = NA
summary(as.vector(sstC))
```

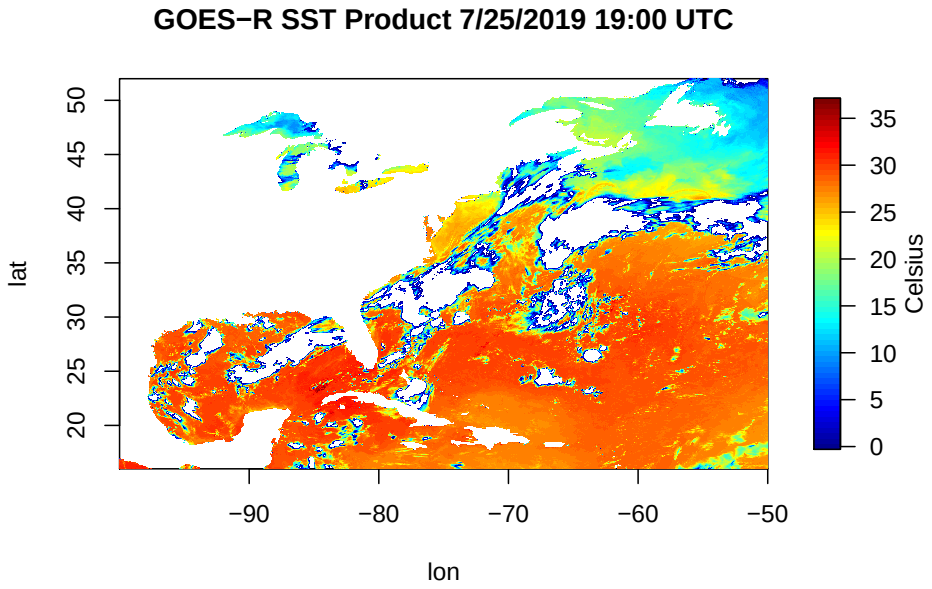
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##     0.0    19.8    27.6    24.1    29.0    36.8 2679819
```

Now we see that our summary looks good (even though we're looking at a matrix in celsius now). So all that's left to do is plot this up...

```
# James Simkins
# Load libraries
library(ncdf4)
library(fields)
ncFile <- ncdf4::nc_open("~/Documents/Github/geog473-673/datasets/OR_ABI-L2-SSTF-M3_G16_s20192081
sstK <- ncdf4::ncvar_get(nc=ncFile, varid="SST")
lat <- ncdf4::ncvar_get(nc=ncFile, varid="latitude")
lon <- ncdf4::ncvar_get(nc=ncFile, varid="longitude")

# convert sst from Kelvin to Celsius
sstC <- sstK - 273.15
# remove values below OC
sstC[sstC < 0] = NA

########################################
# Plot the matrix
fields::image.plot(x=lon, y=lat, z=sstC, legend.lab="Celsius")
title("GOES-R SST Product 7/25/2019 19:00 UTC")
```

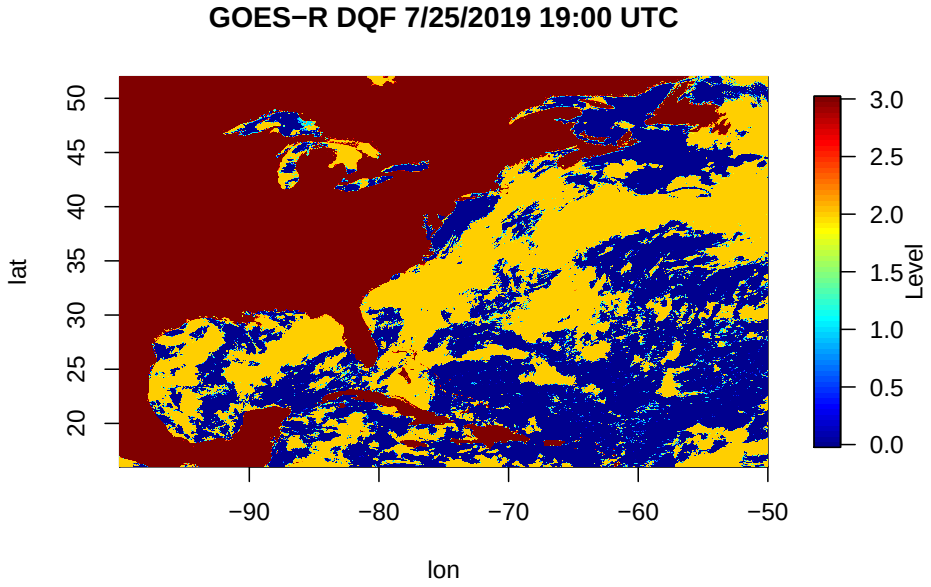


Now we plot this up using the 'image.plot()' function from the 'fields' library. We just tell it to plot the sstC but notice that I didn't place an argument in here. If you don't tell image.plot() what an argument actually is, it assumes you're following the order that the function was written in. If you declare arguments, such as 'z=sstC' or like 'varid="SST" ' from above, you can place the arguments in whatever order you want in that function. Notice that for this kind of plot, the 'title()' function is a separate function rather than an argument of 'image.plot()'. This goes for a lot of plot aesthetics that we'll get into later. Remember, if you're ever confused about something like this just tell R you need 'help()'.

2.3 In Class Exercise

1) Go to https://github.com/jsimkins2/geog473-673/tree/master/datasets
2) Download 'OR_ABI-L2-SSTF-M3_G16_s20192081300453_e20192081400161_c20192081406297.n
3) Make simple plot of the 'DQF' (Data Quality Flag) variable

-Hint: You may have to use 'install.packages' in your console first



Aim for something like this:

2.4 DataTypes

If you look into your environment from the in class exercise, you'll notice under the 'Data' tab you have a large matrix of the DQF values. R stores these matrices without the column/row identifier (aka lat & lon). We provide the image.plot() function with the lon & lat arrays becuase it doesn't know what the x & y coordinates are of the matrix. Notice that the environment tells you the dimensions - the matrix is 2778 x 1989. R is indexed from 1 to the length of the dimension. Here is what I mean:

- `dim(sstC)` is `2778 1989`
- `dim(lat)` is `1989`
- `lat[0]` is `numeric(0)`
- `lat[1]` is `16.00283`
- `lat[1989]` is `51.98563`
- `lat[1990]` is `NA`

The above tests are referred to as indexing. The 1st point of the lat array is 16.00283. In R, we index using brackets []. If you want to find more values other than just a single point, the procedure is referred to as slicing.

`lat[1:10]` is `16.00283 16.02093 16.03903 16.05713 16.07523 16.09333 16.11143 16.12953 16.14763 16.16573`

The lat object we've been exploring here is an 'array'.

An array is a vector with one or more dimensions. So, an array with one

dimension is (almost) the same as a vector. An array with two dimensions is (almost) the same as a matrix. An array with three or more dimensions is an n-dimensional array.

A vector is what is called an array in all other programming languages except R — a collection of cells with a fixed size where all cells hold the same type (integers or characters or reals or whatever).

A list can hold items of different types and the list size can be increased on the fly. List contents can be accessed either by index (like mylist[[1]]) or by name (like mylist$age).

```
mylist = list("delaware", "pennsylvania", "maryland")
mylist[[1]]
```

```
## [1] "delaware"
```

A matrix is a two-dimensional vector (fixed size, all cell types the same).

A data frame is called a table in most languages. Each column holds the same type, and the columns can have header names.

```
# load in base dataset
data(mtcars)
mtcars
```

```
##                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
```

```
## AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28        13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird   19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9         27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2      26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa       30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L     15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino       19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora      15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Notice that it's organized just like an excel spreadsheet. In essence, a dataframe in R is just an advanced Excel Spreadsheet.

```
class(mtcars)
```

```
## [1] "data.frame"
```

Let's explore a column of this data:

```
mtcars$mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

Now let's sort the entire dataset by this column from least to greatest

```
mtcars[order(mtcars$mpg),]
```

```
##                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Camaro Z28         13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## AMC Javelin        15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Merc 280           19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Ferrari Dino       19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
```

```
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag      21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Volvo 142E         21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Toyota Corona      21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Porsche 914-2      26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Fiat X1-9          27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Honda Civic        30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Lotus Europa       30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Fiat 128           32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Toyota Corolla     33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
```

Now let's talk details.