Magneto - Scale and Standardization Project

Includes:

Final Report

By: Robyn Hawthorne

Completed for: Natural Resources Canada

Supervising Professor: Dr. Wesley Burr

Trent Community Research Centre Project Coordinator: Brittany Finigan

Course Code: MATH 4851

Course Name: Community-Based Research Project (Sc)

Completion Date: April, 2023

Project ID: 6102

Magneto Scale and Standardization Project

Robyn Hawthorne
Department of Mathematics
MATH 4851H
Supervised by Dr. Wesley Burr
December 2022

A Community-Based Research Project Completed in Partnership with
Natural Resources Canada
And Coordinated by the Trent Community Research Centre

Contents

Overview

The “magneto” project has been an ongoing project with many parts to it which has been worked on by previous students at both Trent and Queen’s. I worked on the digitization aspect of this project. The goal of my project was to write a code which, when applied to all the files from the magnetogram machine, would run and create websites holding the data from each day of magnetogram readings.

For these websites to be available to the public to be used in research there needed to be a way for these webpages to be all in one place. For that, we created two more code files; one to organize the files by year and another to organize each years’ files by days.

Magneto data

The data files reflect magnetogram readings for Ontario and were recorded and provided by the Natural Resource Canada Geomagnetic Laboratory. The data recorded by the magnetogram reflects variation in the earth’s geomagnetic field at a given time. Knowing the level of the magnetic field can be extremely useful to help researchers understand a range of phenomena from weather events such as solar flares to mapping of the ocean floor. The work done in this project will allow the data to be universally available for use by external research teams.

Use of R and HTML

To create the algorithms necessary, I wrote my code in R studios and would then source the code into an HTML format which is in itself a webpage. I’ve been using R for a few years now so I am well versed in functions I can use to get my desired outcome. Before starting on this project, I was not familiar with HTML, however.

My first task in the project was to understand the necessary parts that must be included in an HTML document for it to work properly, such as the header and footer. The final lines of code in my various files were used to create the HTML, similar to the example provided below:

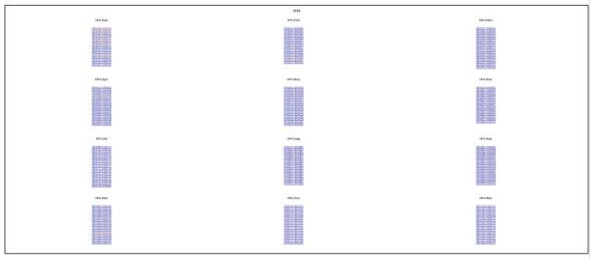
```
html <- write(x = c(top, "<body>", html_block, "</body>", bottom))
```

Another important part to the webpages which required me to do some research in HTML coding was creating divs tags, which help format the layout of a webpage. They create blocks where where information can be input, stored, and contained within the webpage layout. There can be many divs in one webpage and the layout of these can be altered using more code (which I will get into more when I talk about my experience coding the parent/index webpages). Because of the desired usage of the webpages, the information had to be organized and thus creating multiple divs for my webpages was a necessary task, though there was a different layout for each webpage based on the data included in the page.

HTML logic requires certain lines of code to be included in the ‘html_block’ portion (which was usually the div block coding). The div blocks required certain lines of code to run properly. An example of this is included below. A function was made in the R code to create these div blocks.

```
str1 <- "<div class='myDiv'>"
    str_link_start <- "<a href='../Storage/" str_middle
    <- "'>" str_link_end <- "</a></br>"
```

Figure 1: Examples of div blocks organizing data on the website.



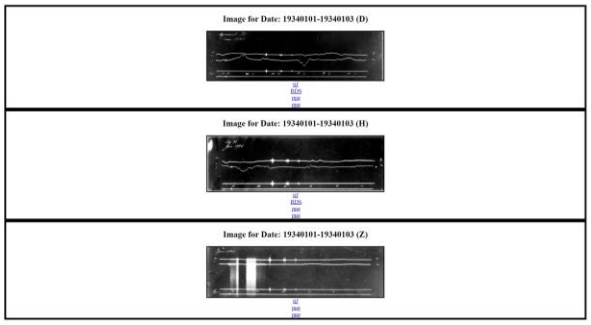


Image for Date: 19340101-19340103 (D)
Image for Date: 19340101-19340103 (H)
Image for Date: 19340101-19340103 (Z)

Keeping these div blocks above organized required some experimentation with the size of the divs blocks. The desired size would change based on the page I was making for the type of organization it would require, and certain pages have much more within the div blocks than others.

The algorithm made to create the websites could have been done using various programs. I used R because I am familiar with its logic and there was some pre-existing work when I began that was already in an R file for me to build upon.

Working with the files

At the beginning of the project, I had a 8GB USB stick holding a small portion of the files so I could begin to experiment with them and to understand what was to be done. The data included a year's worth of magnetogram observation data and the beginnings of a code to create the webpages.

Pre-existing R code

The magneto project has been worked on by many previous students. One of these students worked to create a calendar for the webpage that would allow users to retrieve data from their desired dates. For this code to be develoeped, a draft of the website's layout and functionality was needed. The code for the webpages, created as a draft for this previous project, not fully functional, and was unable toattach the proper documents. However it would create a website for each day - though many had no information attached to said webpage within a single div using the following code:

```
make_div <- function(in_files, date) { stopifnot(length(in_files) >= 1)
    str1 <- "<div class='myDiv'>" str2a <-
    "<h2>Image for Date: " str2b <- "</h2>"
    str2 <- paste0(str2a, date, str2b)
    str4a <- "<a href='../Storage/" str4b <- "'><img
    src='../Storage/" str4c <- "'
    style='width:500px'></br>"
    # grab last 7 digits

    last7 <- sapply(in_files, FUN = function(x) { n <- nchar(x); substr(x, (n-6), n) }) id <- which(last7 == "tif.png")
    # insert the png as absolute ref here str4 <- paste0(str4a, in_files[id], str4b, in_files[id], str4c)
    # grab actual file terminators

    in_files_type <- sapply(in_files, FUN = function(x) { n <- nchar(x); substr(x, (n-2), n
    str_link_start <- "<a href='../Storage/" str_middle
    <- "'>" str_link_end <- "</a></br>"
    links <- vector(length = length(in_files))
    for(j in 1:length(in_files)) { links[j] <- paste0(str_link_start, in_files[j], str_middle, in_files_type[j],
        str_link_end)
    } links <- paste0(links, collapse = "&nbsp;")
    str5 <- "</div>"
    html_block <- paste0(str1, str2, str4, links, str5)
    # note: put these in a data.frame, then do a do.call() or similar
    return(html_block)
}
```

The previous code was also able to create the names for each of the webpage specific to the days in which the data represented:

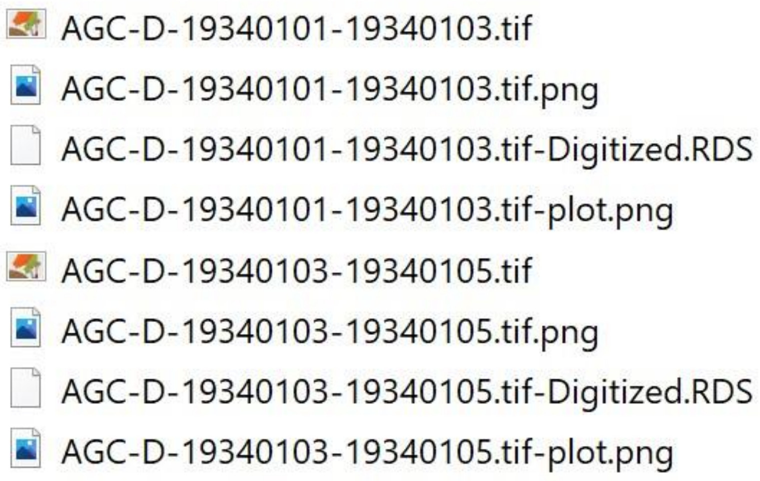
```
s_date <- unlist(lapply(fname_split, FUN = function(x) { substr(x[1], 7, 14) })) s_year <-
unlist(lapply(fname_split, FUN = function(x) { substr(x[1], 7, 10) })) s_month <- unlist(lapply(fname_split, FUN =
function(x) { substr(x[1], 11, 12) })) s_day <- unlist(lapply(fname_split, FUN = function(x) { substr(x[1], 13, 14)
}))
e_date <- unlist(lapply(fname_split, FUN = function(x) { substr(x[1], 16, 23) })) e_year <-
unlist(lapply(fname_split, FUN = function(x) { substr(x[1], 16, 19) })) e_month <- unlist(lapply(fname_split, FUN
= function(x) { substr(x[1], 20, 21) })) e_day <- unlist(lapply(fname_split, FUN = function(x) { substr(x[1], 22, 23)
}))
```

This code however could not properly or fully create the webpages so that each had functioning links to the data files for the corresponding day. This is because of some missing logic to take into account that the data has multiple readings for each of the days.

Files to be Linked on the Webpages

There were many data files from the magnetogram which I was given on the USB drive. I was the full years' worth of data from the year 1934, however had more than 365 files. The data was recorded in three-day chunks, i.e., my first observation file was labelled 19340101-19340103. As stated above, there were also multiple readings for each day due to there being more than one machine measuring the data (one specified as D, one as H and one as Z).

The file for each of these specific machines by the day of the observation also had multiple files that would need to be attached to the webpage. Each had a '.tif' file, a 'tif.png' file, a 'tif-Digitized.RDS' file, and a 'tif-plot.png' file. This meant that each days' webpage could have four different files to be linked to so they could be downloaded.



AGC-D-19340101-19340103.tif
AGC-D-19340101-19340103.tif.png
AGC-D-19340101-19340103.tif-Digitized.RDS
AGC-D-19340101-19340103.tif-plot.png
AGC-D-19340103-19340105.tif
AGC-D-19340103-19340105.tif.png
AGC-D-19340103-19340105.tif-Digitized.RDS
AGC-D-19340103-19340105.tif-plot.png

Creating individual webpages

As mentioned above, the coding for this task was partially started, though not complete. The goal was to provide a link to all the files you would need for a particular date, and to print out the 'tif.png' picture first. This would give a user an overview of the data from that day with the option to download the other files after.

Daily data

The files which were to be included in each webpage needed to be included in the coding for the div so that it was included in the layout of the webpage. This was completed using the following code:

```
sub_files_type <- sapply(sub_files, FUN = function(x) { n <- nchar(x); substr(x, (n-2), n
for(j in 1:length(sub_files)) { links[j] <- paste0(str_link_start, sub_files[j], str_middle, sub_files_type[j],
   str_link_end)
   }
links <- paste0(links, collapse = "&nbsp;")
divs[k] <- paste0(str1, str2, str4, links, str5, collapse = "&nbsp;")
```

We are able to subset our total files for the ones with each unique file ending (i.e. the tif vs. RDS vs. png,...) and ensure that each one will show up once we run this through all the dates.

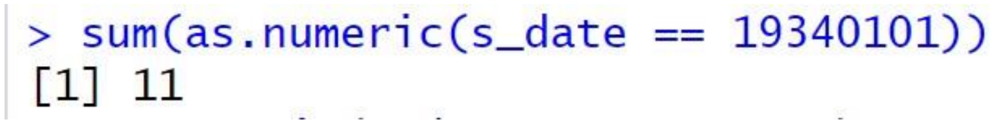
Getting the ‘tif.png’ file to print for each date in the webpage requires a bit more work. Since we want the image to show up on the layout, we must also include this in the div code before we iterate it through each of the days. For this we use the following:

```
# grab last 7 digits last7 <- sapply(sub_files, FUN = function(x) { n <- nchar(x); substr(x, (n-6), n) }) id <-
    which(last7 == "tif.png")
    # insert the png as absolute ref here
    str4 <- paste0(str4a, sub_files[id], str4b, sub_files[id], str4c)
```

As seen in prior coding, we then paste ‘str4’ in the output for the ‘make_div’ function.

Various machine readings

Even after taking these measures there was still something slightly off with the links in the webpage. The links did not bring you to the proper files, so there needed to be some investigating. Through further research and experimentation, I learned that the way we subset the data looking at a specific date range returned the following result:



```
> sum(as.numeric(s_date == 19340101))
[1] 11
```

This indicated that there were 11 files under the same date. However, we know that there should be, at most, four different files for each date. This meant that the code was recognizing at least three different files for each file type on each day.

Looking closer at some of the file names, it was noticed that the fifth character in the files names was not always the same. There were in fact three possible options for that character; D, H, Z. This is to differentiate the observations coming from each machine. Recall that the goal is to have all the data for each day in one webpage, that means each of the D, H and Z observation data should be in the webpage for each specified date.

This problem can be solved by including some logic for this in the ‘make_div’ function again. If each webpage has a new div block for each of the unique machines, then by iterating our ‘make_div’ function through these unique character values we can have multiple blocks layout in the webpage for each of these observations.

```
#create unique div blocks by 5th digit - still shared webpage/html, but
# with separate labeled blocks by axis
fifth <- sapply(in_files, FUN = function(x) { n <- nchar(x); substr(x, 10, 10) }) str3 <- unique(unlist(fifth))
```

Note: instead of using a hard coded three here to iterate through, we use the length of unique vector value in a situation where there may be more or less than three we are recording data from.

```
divs <- vector(length = length(str3))
for(k in 1:length(str3)) {
sub_files <- in_files[which(substr(in_files, 10, 10) == str3[k])]
}
```

With this now sorted, the rest of the logic in the code was the same, resulting in multiple divs in each of our webpages based on that fifth element for each date. Putting it all together we received the website output shown in Figure 2, for each of the days there are files for.

Figure 2: Website output for a single day os magnetogram data.

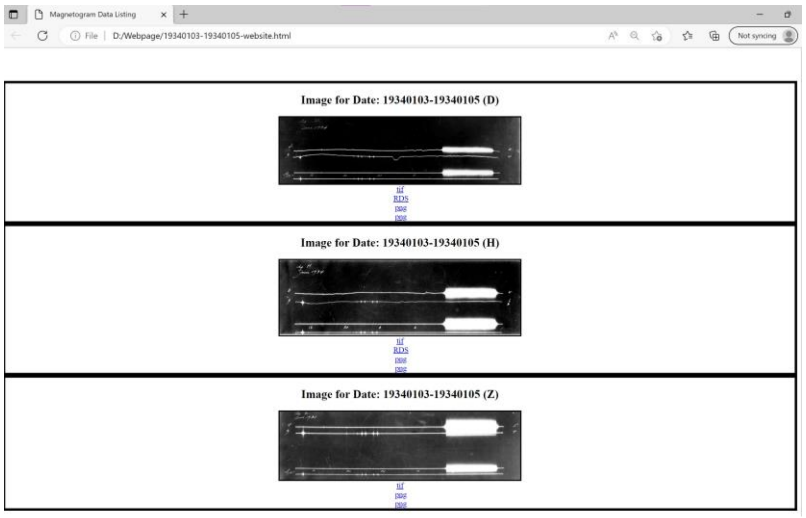


Image for Date: 19340103-19340105 (D)

tif
RDS
png
png

Image for Date: 19340103-19340105 (H)

tif
RDS
png
png

Image for Date: 19340103-19340105 (Z)

tif
png
png

Creating the parent pages

Now that there are functioning websites holding the magnetogram data for each day, we wished to simplify this so that all these links could be found in one place. Thus, I was tasked with creating a 'parent' or 'index' page to hold all the links to the webpages which hold the magnetogram data.

There were 181 webpages created for just the data from the year 1934, so it was decided that keeping all the webpages together in one large index would be overwhelming. Though users could use the 'Ctrl + F' function to find a specific date, it would be a lot to look at considering how many years' worth of data there is. It is for this reason we made an index page for each year's data, followed by an index page which would encompass the webpages for each year.

Years

As mentioned above, there were 181 webpages created for just a year's worth of data. Simply placing the links together one under the other would still be hard to read because of their sheer number. An easy way to layout the yearly index pages would be to order them by months. To do this, we must create divs for each month so that only one month's worth of links appears in any given div block.

Using a similar strategy as used to create the multiple div blocks for each date, we need to use logic for creating 12 divs within our 'make_div' function. Blocking by months is a less complex task than other div blocking because there are consistently 12 months to a year. Knowing the months allowed us to use certain R packages, as follows, to give each div block a label by month.

```r
month_names <- levels(lubridate::month(paste0("1901-", 1:12, "-01"), label = TRUE))
```

Keeping some logic similar to that from before, the months were measured by the length of their unique values to help in the event that some years were missing a month of data. However, there can only be a maximum of 12 months in the year, and in the case a date was entered incorrectly, code was factored in to prevent errors. Another concern was the possible situation where data has a different starting month from the ending month. The following code was used to prevent these errors:

```r
month <- sapply(in_files, FUN = function(x) { n <- nchar(x); substr(x, 5, 6) }) month2 <- sapply(in_files, FUN =

function(x) { n <- nchar(x); substr(x, 14, 15) }) strM <- unique(c(unlist(month), unlist(month2)))
   if(length(strM) > 12) { stop("Something wrong with your input files.") }
```

The div blocks for month files require more work due to the layout. Without changing anything they simply print one below the other, which still requires users to do a lot of scrolling to find the desired date. By having the blocks float and by creating a 3x4 grid with the months, all the dates could be seen together on one page. To make this happen the dimensions of the blocks had to be adjusted, using the following code:

```
str_div_title <- 0 month_links <- 0
    divs[as.integer(strM[k])] <- paste("<div style='float: left; width:33%;'>",
                                                                            str_div_title,
                                                                        paste(month_links, collapse = ""),
                                                                        "</div>", sep =
                                                                        "")
```

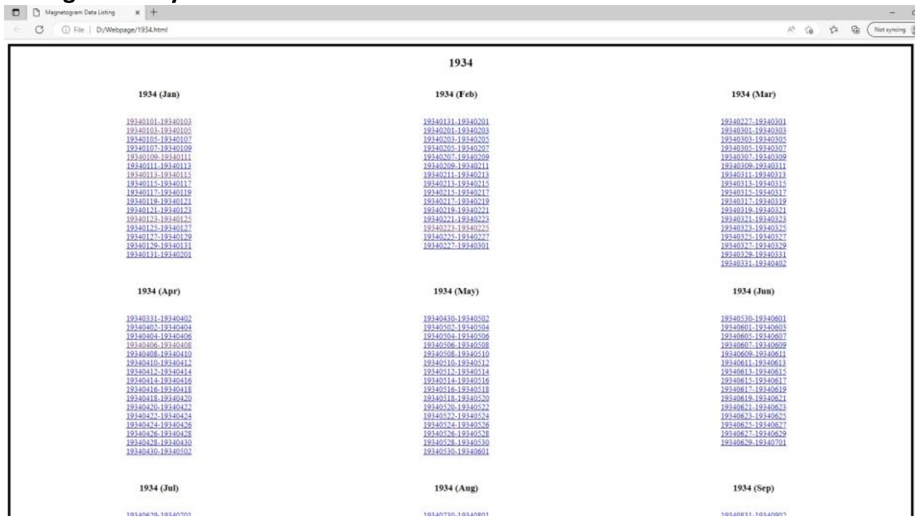
The div blocks also needed to be the same length to maintain organization of the blocks. This was achieved in with the following code:

```
    div_blocks <- vector(length = 4) for(j in 1:4) { div_blocks[j] <- paste("<div style =
    'width:100%; height: 400px;'>",
                                                          paste(divs[(3*(j-1)+1):(3*j)], sep = "",
                                                          collapse = ""),
                                                          "</div>")
    }
```

This formatting resulted in the layout shown in Figure 3, obtained through the following code:

```
im6 <- load.image("monthly.png")
plot(im6, axes=FALSE)
```

Figure 3: Annual data organized by month.



Magnetogram Data Listing

D:/Webpage/1934.html

1934

1934 (Jan)

19340101-19340103
19340103-19340105
19340105-19340107
19340107-19340109
19340109-19340111
19340111-19340113
19340113-19340115
19340115-19340117
19340117-19340119
19340119-19340121
19340121-19340123
19340123-19340125
19340125-19340127
19340127-19340129
19340129-19340131
19340131-19340201

1934 (Feb)

19340131-19340201
19340201-19340203
19340203-19340205
19340205-19340207
19340207-19340209
19340209-19340211
19340211-19340213
19340213-19340215
19340215-19340217
19340217-19340219
19340219-19340221
19340221-19340223
19340223-19340225
19340225-19340227
19340227-19340301

1934 (Mar)

19340227-19340301
19340301-19340303
19340303-19340305
19340305-19340307
19340307-19340309
19340309-19340311
19340311-19340313
19340313-19340315
19340315-19340317
19340317-19340319
19340319-19340321
19340321-19340323
19340323-19340325
19340325-19340327
19340327-19340329
19340329-19340331
19340331-19340402

1934 (Apr)

19340331-19340402
19340402-19340404
19340404-19340406
19340406-19340408
19340408-19340410
19340410-19340412
19340412-19340414
19340414-19340416
19340416-19340418
19340418-19340420
19340420-19340422
19340422-19340424
19340424-19340426
19340426-19340428
19340428-19340430
19340430-19340502

1934 (May)

19340430-19340502
19340502-19340504
19340504-19340506
19340506-19340508
19340508-19340510
19340510-19340512
19340512-19340514
19340514-19340516
19340516-19340518
19340518-19340520
19340520-19340522
19340522-19340524
19340524-19340526
19340526-19340528
19340528-19340530
19340530-19340601

1934 (Jun)

19340530-19340601
19340601-19340603
19340605-19340607
19340607-19340609
19340609-19340611
19340611-19340613
19340613-19340615
19340615-19340617
19340617-19340619
19340619-19340621
19340621-19340623
19340623-19340625
19340625-19340627
19340627-19340629
19340629-19340701

1934 (Jul)

19340629-19340701

1934 (Aug)

19340730-19340801

1934 (Sep)

19340831-19340902

All available years

Now that there is functioning code to create a page containing the webpage files for each year’s data, there needs to be another page where all of those webpages are found. To do so, a new R file was created to run the new code.

Again, simply placing each link below one another for each year would leave a lot of white space in the webpage and also be cumbersome to any user. The goal was to create div blocks by decade so each year is held in a div block with nine other links from that same decade, laid out in a 3x n construction (where n represents the number of years in total, increasing continually as more data is collected).

This code again used a similar structure to that for the index page for each year. We first had to subset the webpage file, to only those for each year, not the daily files. After that we needed to separate the data by the decade. To do so we used the following code:

```
divs <- rep("", length(in_files)) modu <-
as.numeric(year) %% 10
decades <- unique(as.numeric(year) - modu)
decade_map <- as.numeric(year) - (as.numeric(year) %% 10)
```

The object ‘year’ represents the unique years within the files we are selecting from. Thus, this code takes the years and put them into ‘modu’ which writes each year under mod 10. That is to say, it will be 0,1,...,9 by the last digit of the year. Subtracting that from the original year, we receive the years only as the first year of each decade (i.e. for this year it would be 2020). Finding the unique decades, we now know how many div blocks we must create. This was done using the following code:

```
 for(k in 1:length(decades)) { dec_files <- in_files[decade_map
    == decades[k]]
    dec_links <- paste(str_link_start,
                         dec_files, str_middle,
                         substr(dec_files, 1, 4),
                         str_link_end, sep = ""
    )
    # this is the individual years divs[idx] <- paste("<div style='float: left; width:33%;'>",
    "<h3>", decades[k], "s</h3></br>", paste(dec_links, collapse = ""),
                                              "</div>", sep =
                                              "")
    idx <- idx + 1
 }
```

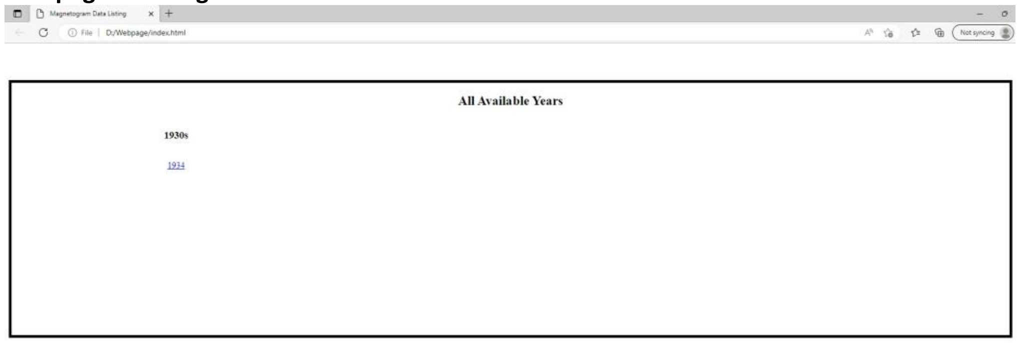
In this case, as the total number of years will continue to change as data continues to be collected, we use the following code to account for changing volumes of data:

```
div_blocks <- vector(length = ceiling(n_dec / 3)) for(j in 1:length(div_blocks)) {
div_blocks[j] <- paste("<div style = 'width:100%; height: 400px;'>", paste(divs[(3*(j-
1)+1):(3*j)],
                                                     sep = "", collapse =
                                                     ""),
                                             "</div>")
}
```

By taking the ceiling after dividing by three, it allows all data from the decade to print properly. When data is not divisible by three, it simply leaves space beside it for the coming decade. When sourced, we receive our final webpage using the following code, and as shown in Figure 4.

```
im7 <- load.image("yearfiles.png")
plot(im7, axes=FALSE)
```

Figure 4: Final webpage allowing users to access all annual data from 1934.



Note: I only had 1934's data, and so this does not accurately represent how this page will look when it is complete. However the logic should hold for all the files as they are run through the previous code files and create an output similar to that shown in Figure 3.

Conclusion

This project required me to learn about HTML coding and what is required in the construction of functioning webpages. The project also required a lot of subsetting of the data and creating functioning for loops that would allow the code to iterate through the files. I was using more applied statistics than I have used before in past courses.

Sorting through 1934 files of magnetogram data, creating 183 webpages, and writing three R files with algorithms to create those webpages from the files, my project is done. Next steps include uploading these files to the server with all the data, and creating the webpages for all the files which exist.