R Introduction Part II

Dr. Robert Buscaglia

October 30, 2018

Further R

You have been introduced to the basics of R, and using R as a calculator. Successful completion of your assignment has also made you aware of the capabilities of making formal documents using R markdown.

Today, we want to dive deeper into some of the ideas that make R stand out as a successful programming language. The goal of this document, and of the second assignment, are to familiarize you with.

- Data Frames
- Useful Packages and tidyverse
- Data Importing and Manipulation
- Graphics
- Control Schemes

This is a lot of content to cover in a single session, so everything is abbreviated. For further studies, or hep with your assignment, I suggest consulting the NAU textbook. The link has been provided to you on the course website, and is given below.

https://dereksonderegger.github.io/570L/

Data Frames

Last week we briefly introduced the concept of a data frame. This storage structure is part of what put R on the map as an important programming language. It allows us to combine all other types of storage into a single structure that acts much like a matrix. If using only numerical data, it is best to stick with matrices. However, it is rare that we will work with only numerical results . We are frequently given additional information in the form of lists, strings, or even logicals. Although we can convert all of these structures to a numerical form, information is often lost.

Data Importing and Pathing

Let's view an example of where we might want to view information in a different form that just numerical results. We can also introduce different formats for importing data. Data can be imported in nearly every available form. Built-in to R is the ability to read in basic files such as .txt and .csv. We can read in files using the *read.table()* commands.

```
read.table("My_File.txt")
read.csv("My_File.csv")
```

These commands also come with many options for changing the way data is read. This includes how to deal with headers, missing data, how columns and rows should be named, quotations, skipping lines, and much more. Remember, we can also learn more about a function using the "?" command!

```
?read.csv
```

There are also packages available for reading in nearly all other files types not provided through base R. One major one is reading data from Excel files. This requires the package *readxl*, and provides the function *read_excel()* with many of the same options as importing data from base R.

Pathing

What is critical when attempting to import data is that you tell R where the data is located. Sometimes, this means downloading data from a website. Most of the time, this means locating the data on your computer.

Let's read in the BodyFat data set using the online resource from Lock5Data. I use the *read.table()* commands with options for .csv. Although, we could have also used *read.csv()*. I then give it the URL for where the data is found online! It will download the data and create the data frame for me. I then use the structure commands, *str()*, to evaluate what the data frame contains.

```
BodyFat <- read.table(file = 'http://www.lock5stat.com/datasets/BodyFat.csv',
header = TRUE, sep=",")

str(BodyFat)
```

```
## 'data.frame':    100 obs. of  10 variables:
##  $ Bodyfat: num  32.3 22.5 22 12.3 20.5 22.6 28.7 21.3 29.9 21.3 ...
##  $ Age    : int  41 31 42 23 46 54 43 42 37 41 ...
##  $ Weight : num  247 177 156 154 177 ...
##  $ Height : num  73.5 71.5 69 67.8 70 ...
##  $ Neck   : num  42.1 36.2 35.5 36.2 37.2 39.9 37.9 35.3 42.1 39.8 ...
##  $ Chest  : num  117 101.1 97.8 93.1 99.7 ...
##  $ Abdomen: num  115.6 92.4 86 85.2 95.6 ...
##  $ Ankle  : num  26.3 24.6 24 21.9 22.5 22 23.7 21.9 24.8 25.2 ...
##  $ Biceps : num  37.3 30.1 31.2 32 29.1 35.9 32.1 30.7 34.4 37.5 ...
##  $ Wrist  : num  19.7 18.2 17.4 17.1 17.7 18.9 18.7 17.4 18.4 18.7 ...
```

What if this file is instead located on your computer? We would need to either change our working directory (*setwd()*) or give it the full path for where the file can be located. Below is an example, but will likely not work for your computer, as the paths will be different.

```
### Change working directory
setwd("C:\\Users\\RBuscag\\Desktop")
read.csv("BodyFat.csv")

### Give the absolute path
read.csv("C:\\Users\\RBuscag\\Desktop\\BodyFat.csv")
```

This gives two alternative methods for loading in data. Remember, we often also want to load data from packages, which we learned how to do last time!

Manipulating a Data Frame

We should now have a solid understanding of how to import data from any source. In most cases, we will obtain our data in the form of a data frame. Data frames can be manipulated in a variety of ways. Maybe we need to calculate summary statistics, or the data we are interested in is some combination of the results we have been given. We will begin to enter the world of *tidyverse* which is a compendium of packages built to help work with and use data frames. Let's first look at some basic examples using only base R.

Working with Data Frames in Base R

First I will need a data frame to work with. I do not import a data set here, but rather enter one in manually.

```
grades <- data.frame(
  l.name = c('Cox', 'Dorian', 'Kelso', 'Turk'),
  Exam1 = c(93, 89, 80, 70),
  Exam2 = c(98, 70, 82, 85),
  Final = c(96, 85, 81, 92) )
kable(grades, align="c")
```

l.name	Exam1	Exam2	Final
Cox	93	98	96
Dorian	89	70	85
Kelso	80	82	81
Turk	70	85	92

We will call this the *grades* data frame. It includes Exam scores for 4 students. I am interested in calculating the mean score of the three exams for each student. In base R, we do everything through vectorization. Here, I will take the data frame, and reduce it to its numerical components.

```
scores<-grades[,-1]
```

The command above teaches us a lot! * grades is the data frame we are working with * the [] indicates I want to pull information from an object * the first values are rows, the second values are columns (and so on if you have arrays!) - here I entered [,-1], which actually REMOVES only the first column. This is a useful trick! * We can save our new information within another objects, that we call scores

I have the data frame and removed the names, which now lets me work with the data frame as a matrix.

```
kable(scores, align="c")
```

Exam1	Exam2	Final
93	98	96
89	70	85
80	82	81
70	85	92

There are many methods now I could use to calculate the mean of each row. One option are the functions *rowMeans()* and *colMeans()*. Here I am interested in the mean for each student, which means taking the mean of each row.

```
Exam.Ave<-rowMeans(scores)
Exam.Ave
```

```
## [1] 95.66667 81.33333 81.00000 82.33333
```

Another useful function is the *apply()* command. This command can actually allow you to compute faster than many other programming languages. The difficulty of using the *apply()* function is that it requires vectorized calculations; however, upon vectorizing, the speed of *apply()* is bested by almost nothing else, even C and Python!

The *apply()* function requires you to give it a source to work from (data frame, matrix, arrays, there is even an *lapply()* function for lists), tell it which coordinate to work through (1 = rows, 2 = columns, 3 = dimension 3, and so on...). Finally, you must supply it a function to use on each of those objects. Here, we will use the *mean()* function.

```
Exam.Ave<-apply(scores, MARGIN = 1, FUN=mean)
Exam.Ave
```

```
## [1] 95.66667 81.33333 81.00000 82.33333
```

We obtain the same result with either method. How can we now put these scores together with the original data frame? The commands *rbind()* and *cbind()* were introduced last time. We need to only paste these onto our data frame.

```
Exam.Ave<-round(Exam.Ave, 3)
Final.Grades<-cbind(grades, Exam.Ave)
kable(Final.Grades, align="c")
```

l.name	Exam1	Exam2	Final	Exam.Ave
Cox	93	98	96	95.667
Dorian	89	70	85	81.333
Kelso	80	82	81	81.000
Turk	70	85	92	82.333

The tidyverse

tidyverse is more than just a package, its a philosophy! Let's take a moment and evaluate all that has been accomplished by Tidyverse.

https://www.tidyverse.org/

Although I have only just begun my adventure through the tidyverse, I have found there are some remarkable things you can do with the packages that are available. If you aspire to learn data science to any degree, than tidyverse is an excellent collection of packages to master!

Today (and in your assignment) we will focus on only three of the tidyverse packages:

```
library(ggplot2)
library(dplyr)
library(tidyr)
```

The main philosophy behind these packages is improving the work flow of data frames. Let's revisit our grades example. Through the use of dplyr, the work we did before through vectorization/matrix manipulation and combining can be done in one swift flow:

```
Final.Grades<-grades %>% mutate(Exam.Ave=round(rowMeans(.[,2:4]),3))
kable(Final.Grades, align="c")
```

l.name	Exam1	Exam2	Final	Exam.Ave
Cox	93	98	96	95.667
Dorian	89	70	85	81.333
Kelso	80	82	81	81.000
Turk	70	85	92	82.333

The big "idea" here is what is known as piping. The command %>% is read as "pipe". I took the data frame and piped it to the command mutate. Mutate (how would you look up information about this command? ?mutate), states to create a new column, which I name Exam.Ave. I then calculate the rowMeans of the columns of the grades data frame. This is all done using .[,2:4]. The . represents call the data frame we started with, and [,2:4] has been previously explained. I then round the answer to 3 decimals.

That is a lot of work, all done in one brief line! This is the philosophy of tidyverse.

Let's do a real example!

I will import all of the maximum temperatures for Flagstaff.... from 1985 to 2015!!!

```
FlagTemp <- read.csv(
  'https://github.com/dereksonderegger/570L/raw/master/data-raw/FlagMaxTemp.csv',
  header=TRUE, sep=',')
str(FlagTemp)
```

```
## 'data.frame':    365 obs. of  34 variables:
##  $ X    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Year : int  1985 1985 1985 1985 1985 1985 1985 1985 1986 1986 ...
##  $ Month: int  5 6 7 8 9 10 11 12 1 2 ...
##  $ X1   : num  71.1 63 81 77 82.9 ...
##  $ X2   : num  71.1 63 86 68 75 ...
##  $ X3   : num  68 64.9 90 78.1 73.9 ...
##  $ X4   : num  68 60.1 88 80.1 72 ...
##  $ X5   : num  64.9 69.1 91.9 82 66.9 ...
##  $ X6   : num  64 75.9 91.9 81 63 ...
##  $ X7   : num  64 82 89.1 82.9 63 ...
##  $ X8   : num  64.9 86 88 82.9 64 ...
##  $ X9   : num  69.1 84.9 90 80.1 68 ...
##  $ X10  : num  66 84 87.1 80.1 64.9 ...
##  $ X11  : num  51.1 82 84 80.1 68 ...
##  $ X12  : num  55.9 82.9 84 75.9 66 ...
##  $ X13  : num  59 84.9 84.9 79 66.9 ...
##  $ X14  : num  57.9 82.9 87.1 79 75 ...
##  $ X15  : num  66 82.9 84 81 73.9 ...
##  $ X16  : num  66.9 86 84.9 81 73 ...
##  $ X17  : num  66 84.9 79 79 73 ...
##  $ X18  : num  66 87.1 81 75 70 ...
##  $ X19  : num  68 87.1 75.9 82 52 ...
##  $ X20  : num  66.9 84 73.9 82 59 ...
##  $ X21  : num  66.9 81 64.9 70 55 ...
##  $ X22  : num  63 82 72 80.1 68 ...
##  $ X23  : num  NA 84.9 73 84 68 ...
##  $ X24  : num  70 81 80.1 88 71.1 ...
##  $ X25  : num  73.9 73.9 80.1 91 72 ...
##  $ X26  : num  71.1 72 77 84.9 73 ...
##  $ X27  : num  71.1 73 80.1 84 72 ...
##  $ X28  : num  69.1 81 82 81 64.9 ...
##  $ X29  : num  73 84 77 79 61 ...
##  $ X30  : num  69.1 82 73.9 82 64 ...
##  $ X31  : num  62.1 NA 73 82.9 NA ...
```

```
levels(factor(FlagTemp$Year))
```

```
##  [1] "1985" "1986" "1987" "1988" "1989" "1990" "1991" "1992" "1993" "1994"
## [11] "1995" "1996" "1997" "1998" "1999" "2000" "2001" "2002" "2003" "2004"
## [21] "2005" "2006" "2007" "2008" "2009" "2010" "2011" "2012" "2013" "2014"
## [31] "2015"
```

The goal here, is I want to create a data frame that has Months on the columns, Years on the Rows, and gives me the Mean monthly Max Temperature for every month in the data set. There are always multiple ways to tackle things in R. I could use *rowMeans()* and some finessed base R coding, but we can use Tidyverse in a pretty unique way here!

The commands come from the package "tidyr". We can manipulate the way the data frame is structure, and use this to move the information we are interested in around in a smart way! We have to use 11315 observations, so we better be smart about it!

We start with the *gather()* function, which takes the 34 columns of information, and I reduce it to 4 columns of information.

```
FlagTemp.Long<-FlagTemp %>% gather(key=Day, value=TMax, X1:X31)
kable(head(FlagTemp.Long), align="c")
```

X	Year	Month	Day	TMax
1	1985	5	X1	71.06
2	1985	6	X1	62.96
3	1985	7	X1	80.96
4	1985	8	X1	77.00
5	1985	9	X1	82.94
6	1985	10	X1	64.04

We are interested in the average temperature for every month within the observation range. We can use dplyr commands to make this happen in a pretty easy way!

```
FlagTemp.MonthSum<-FlagTemp.Long %>%
  group_by(Year, Month) %>%
  summarise(AveMonthTemp = mean(TMax, na.rm=TRUE))
kable(head(FlagTemp.MonthSum), align="c")
```

Year	Month	AveMonthTemp
1985	5	66.14000
1985	6	79.08800
1985	7	81.76710
1985	8	80.42000
1985	9	67.98800
1985	10	61.20645

Wow, seem confusing? The tidyverse packages are a philosophy! They take some getting used to. This takes the data frame FlagTemp.Long, groups the values by Year/Month, then calculates the mean of the variable TMax for each of those groups. This is exactly what I was after!

Notice I am now very close to my final goal, with only a small amount of work. I need only now take this object and spread it back out into a matrix type form. This is exactly the what the tidyr function *spread()* does for me! *spread()* works much the opposite of *gather()*. I now want to place all of these monthly temperature averages into unique cells, with my each column a month, and each row a year. For the function below "key" will become my columns, and "value" the entries within each cell.

```
Final.Data<-FlagTemp.MonthSum %>% spread(key=Month, value=AveMonthTemp)
kable(Final.Data[,1:10], align="c")
```

Year	1	2	3	4	5	6	7	8	9
1985	NA	NA	NA	NA	66.14000	79.08800	81.76710	80.42000	67.98800
1986	49.76194	45.53857	53.06000	57.27200	67.29742	77.37800	76.25097	78.93935	65.55200
1987	41.85935	41.07714	43.83935	61.16000	64.65548	77.93000	77.98129	76.41355	72.23000
1988	40.30323	46.82207	51.89871	56.42600	65.82839	76.30400	81.54645	76.41355	69.97400
1989	38.41613	43.46857	55.02258	66.49400	68.94065	77.34200	82.91097	77.83032	74.82800
1990	41.06387	41.59786	49.64581	58.24400	64.06323	79.22000	78.16710	75.10129	71.84600
1991	40.58774	50.93214	41.16258	55.39400	63.09355	71.31200	80.43742	79.00323	74.46200
1992	39.51355	44.85448	47.77613	62.74400	64.73677	73.47200	78.23600	76.52387	74.65400
1993	42.02774	41.46286	51.63742	60.27800	68.37161	75.20621	79.19484	76.91290	74.03000
1994	47.45677	41.50143	51.60258	57.60200	63.99935	80.54600	83.07935	81.97613	74.29400
1995	39.19419	50.92571	50.04065	54.19400	59.75484	72.66200	81.87742	81.25032	75.34400
1996	47.60194	51.16690	52.38645	62.48600	72.80194	80.66000	83.25935	81.25032	69.27200
1997	40.17071	43.71385	57.18759	54.54345	72.21548	74.34200	82.49600	78.00200	73.79600
1998	44.10345	39.84500	47.99429	50.16200	NA	72.16483	82.74258	80.54000	71.54000
1999	49.70000	50.48857	56.67200	50.93103	65.32903	75.57800	76.87806	74.37448	68.22800
2000	45.32000	45.70483	47.90207	62.61333	71.23419	75.31571	83.76452	80.02516	76.00690
2001	42.27929	43.24357	51.99161	56.01071	72.65677	79.91000	80.61742	79.48516	78.57800
2002	44.33290	49.32667	51.53290	63.19400	69.73032	82.85600	84.38581	82.24400	73.33400
2003	51.39200	45.61769	49.35800	55.67643	70.39400	78.44000	86.36581	79.68839	76.00400
2004	41.88138	39.65310	58.72129	57.03241	69.18452	78.42800	81.22129	76.79677	72.03800
2005	42.26000	41.38571	48.45448	56.89400	68.46452	75.10400	85.36710	76.45419	73.64828
2006	46.98345	50.35357	45.12839	58.70600	71.76839	82.18400	81.85419	75.89677	69.48200
2007	38.55548	45.51440	55.85871	60.34000	69.63200	80.45750	83.24960	79.49097	73.53200
2008	36.15200	44.26483	51.58516	59.30000	63.25613	79.57400	79.22828	78.93935	73.45400
2009	44.84387	45.28667	52.00903	56.88800	NA	71.42000	83.40452	81.34903	73.46600
2010	41.70258	40.48571	47.38710	54.41000	62.79286	77.66000	80.64645	76.60516	77.55800
2011	44.44323	41.95143	53.60000	58.37310	63.15161	77.73800	81.26000	82.87613	74.26000
2012	46.22000	44.92897	51.79357	61.19600	69.96258	80.89400	79.57226	79.00323	74.27000
2013	41.16258	42.74667	56.90968	61.60400	68.74323	83.01200	79.16000	75.74000	70.38200
2014	48.56600	47.92609	NaN	57.87200	67.65862	79.38200	81.26000	74.64200	73.23200
2015	48.81615	53.87360	56.71786	59.49200	61.51419	79.19600	77.29032	79.72483	76.02800

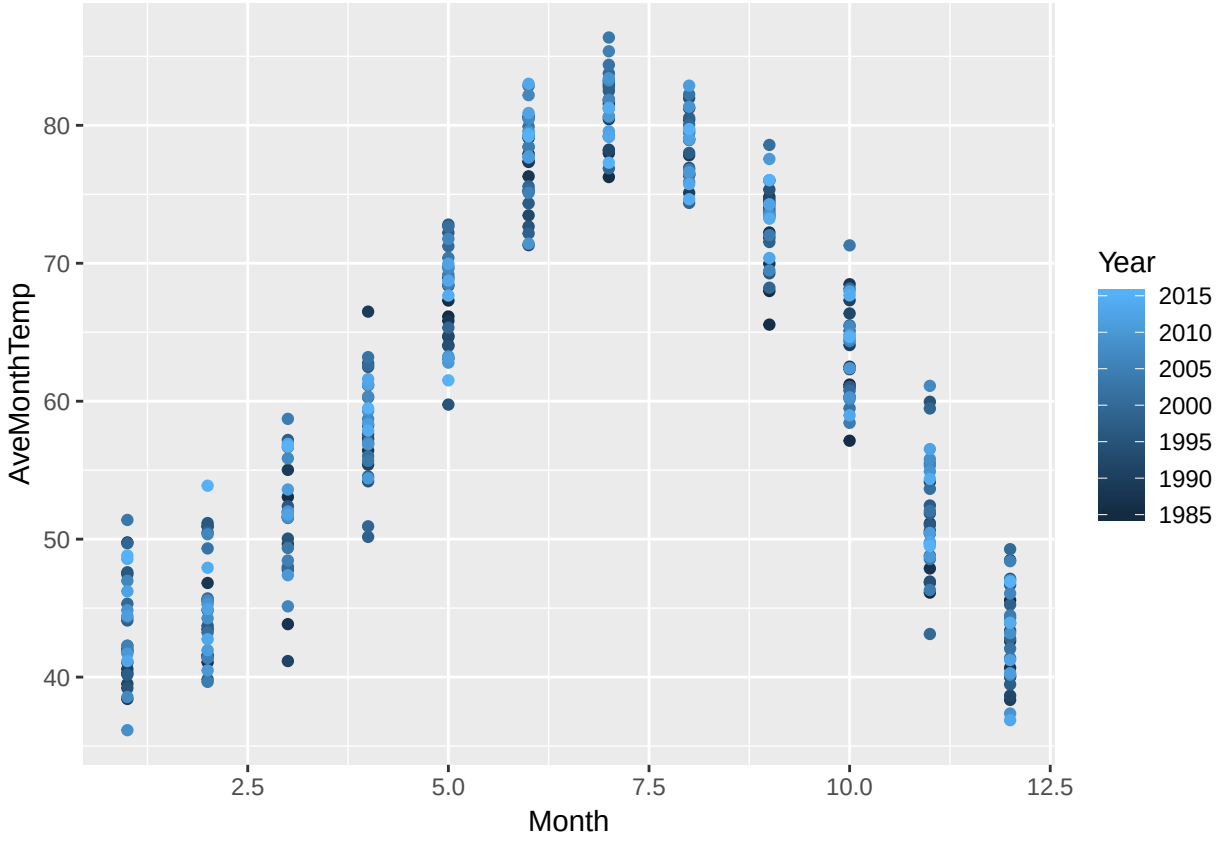
Pretty amazing if you ask me! We just found the mean maximum temperature for every month from 1985 to 2015, and it was a piece of cake!

Visualizing

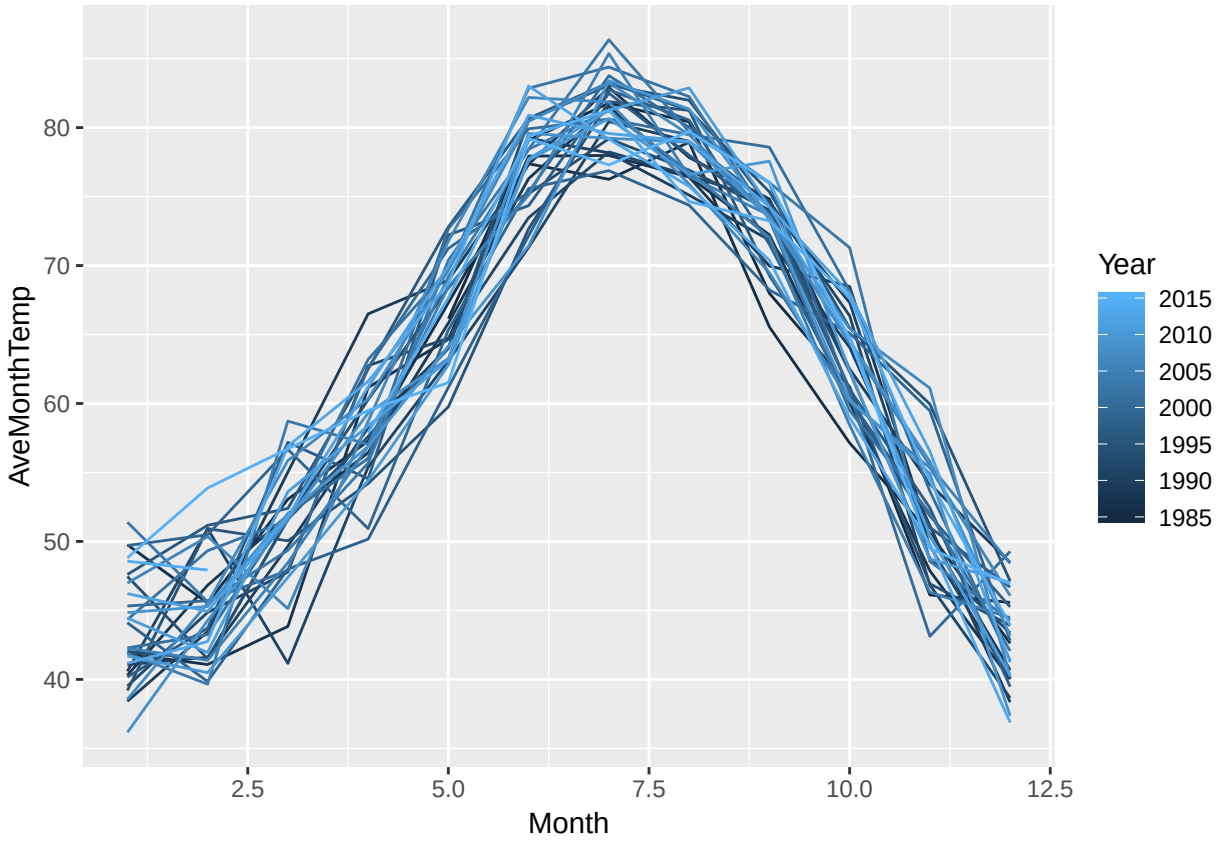
One of the most important aspects of data science is visualizing information. Now that we have the mean temperatures for each month, lets work on visualizing. For a full list of how to interact with ggplot, please consult the online book, Chapters 9 and 10.

Here are four attempts that I made.

```
ggplot(FlagTemp.MonthSum, aes(x=Month, y=AveMonthTemp, color=Year)) +
  geom_point()
```

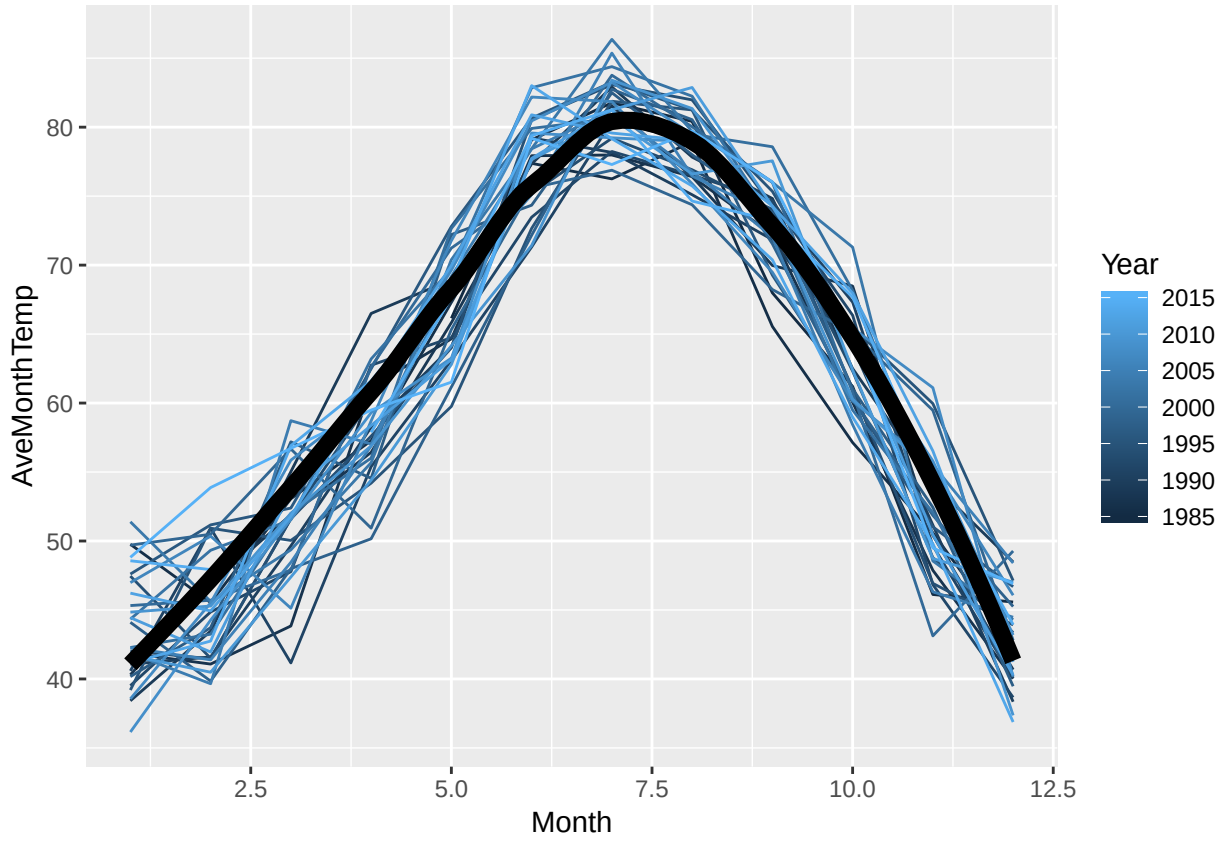


```
ggplot(FlagTemp.MonthSum, aes(x=Month, y=AveMonthTemp, group=Year, color=Year)) +
  geom_line()
```

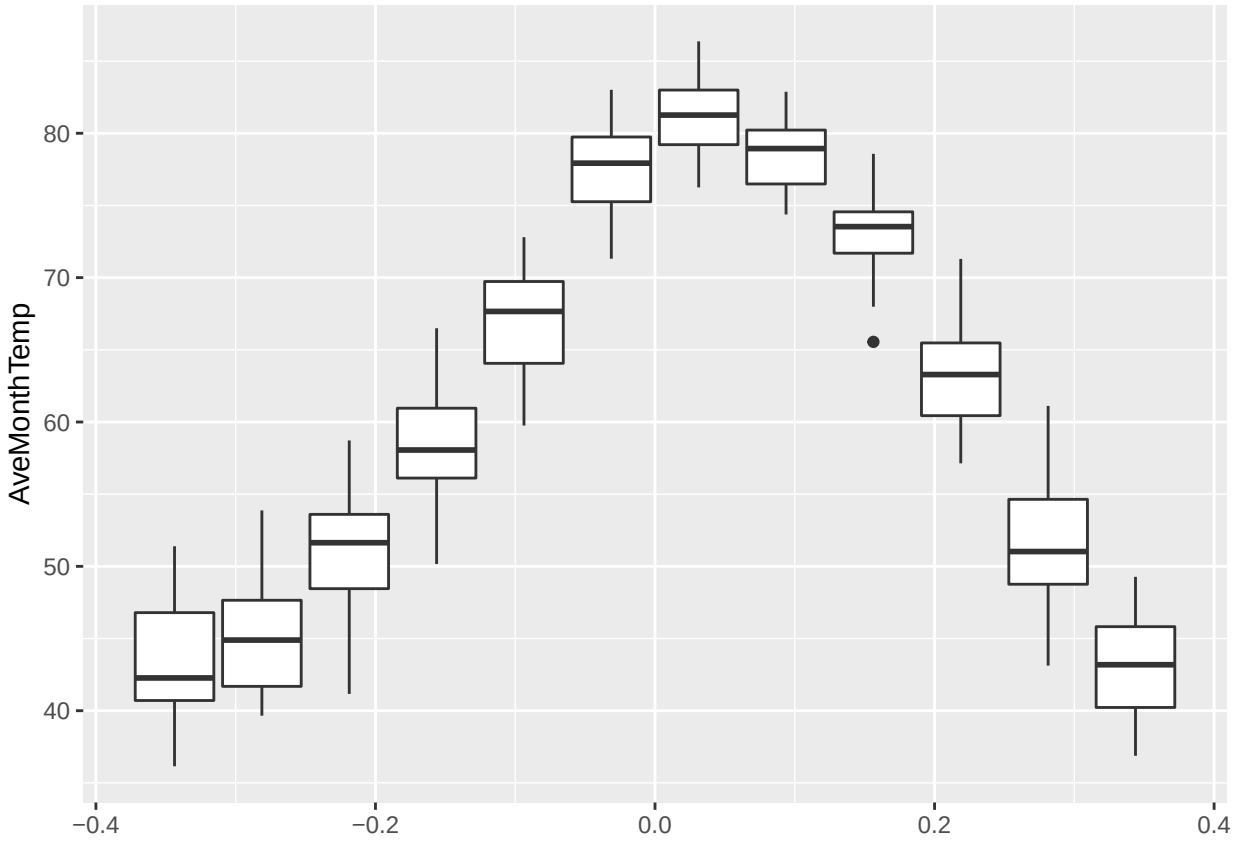


```
ggplot(FlagTemp.MonthSum, aes(x=Month, y=AveMonthTemp)) +
  geom_line(aes(group=Year, color=Year)) +
  geom_smooth(aes(x=Month, y=AveMonthTemp), se=FALSE, color="black", lwd=3)
```

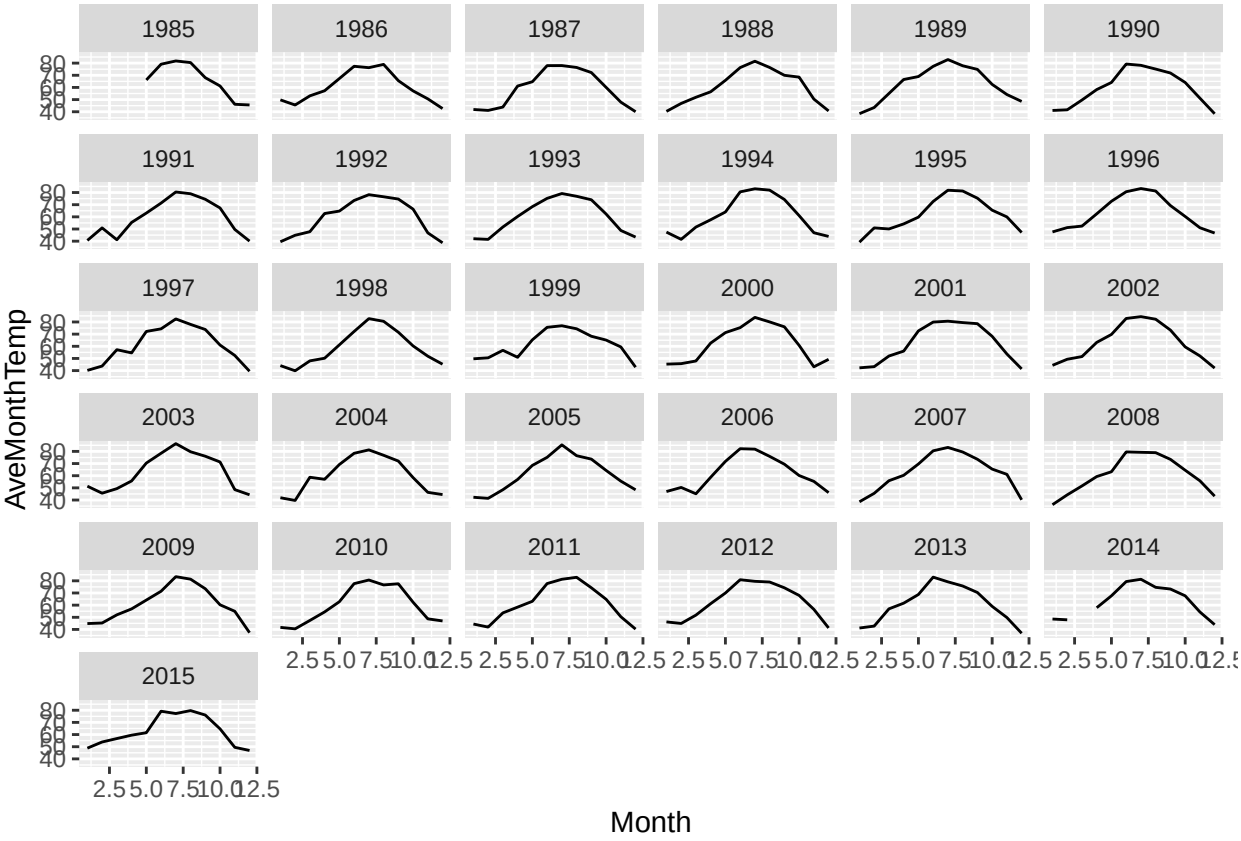
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```
ggplot(FlagTemp.MonthSum, aes(group=Month, y=AveMonthTemp)) +
  geom_boxplot()
```



```
ggplot(FlagTemp.MonthSum, aes(x=Month, y=AveMonthTemp)) +
  geom_line() +
  facet_wrap(~Year)
```



Control Structure

The final topic will be a discussion of control within R. Typically we will want to allow R as a language to the the heavy lifting for us, meaning that rather than evaluating everything by hand, we may wish to evaluate things in loops.

The three major control structures that are used most often in computer science are:

- if-else statements
- for-loops
- while-loops

Let's take a quick look at these three commands.

If-Else

If-Else statements work using logical variables. The idea is that if() will evaluate a logical; if(TRUE) then the command runs. Conversely, if(FALSE), implies do not run! This is often a nice way when developing code to make sure chunks of code run only when you want them to!

We can incorporate the idea of else() to fill in what happens when the if() statement comes back FALSE. Simply, the design structure is: if(FALSE) then do else()

Let's evaluate a PDF using a set of if-else statement. We will work with the following PDF:

$$f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

This simply says, if we are given an x between a and b, then give that value of x the density $1/(b-a)$.

Here is a nested set of if-else that will accomplish this task, given any x.

```
a<-0
b<-9

x<-runif(1, -3, 12)
x
```

```
## [1] 5.393545
```

```
if(x < a){result <- 0}else
  if( x <= b ){result <- 1/(b-a)}else
  {result <- 0}

result
```

```
## [1] 0.1111111
```

The syntax above is fairly sensitive, but it accomplishes the goal!

For-Loops

For-loops allows us to iterative a process multiple times. We can either use the iterator within the loop, or simply just iterate many times. For-loops have the following structure

for(iterator in vector) {

expression 1

expression 2

...

expression n

}

iterator is our object that stores the iteration values. vector is all values for which we wish iterator to take on. We can then run as many expression as we desire within the for loop.

Here is the basic example to show how the iterator works.

```
for(i in c(1, 3, 5, 10))
{
  print(i)
}
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 10
```

We see everywhere that "i" appears, it is replaced by the values within the vector. This can become very useful if we need to do things many times. Like maybe we wish to know the 95th percentile of the t-distribution at every 5 degrees of freedom from 5 to 50!

```
for(df in seq(5, 50, 5))
{
  print(paste("df = ", df, " returns a 95th percentile of ", qt(0.95, df), sep=""))
}
```

```
## [1] "df = 5 returns a 95th percentile of 2.01504837333302"
## [1] "df = 10 returns a 95th percentile of 1.81246112281168"
## [1] "df = 15 returns a 95th percentile of 1.75305035569257"
## [1] "df = 20 returns a 95th percentile of 1.72471824292079"
## [1] "df = 25 returns a 95th percentile of 1.7081407612519"
## [1] "df = 30 returns a 95th percentile of 1.69726088659396"
## [1] "df = 35 returns a 95th percentile of 1.68957245778027"
## [1] "df = 40 returns a 95th percentile of 1.68385101333565"
## [1] "df = 45 returns a 95th percentile of 1.67942739265235"
## [1] "df = 50 returns a 95th percentile of 1.6759050251631"
```

While-Loop

The while-loop works in a different manner than a for-loop. The while-loop works through logicals again, but now, it will continue to loop until the conditional is met. The basic structure is

while(condition) {

Expression 1

...

Expression n

check condition

}

Thus, the while loop needs to be initialized, and every time condition comes back TRUE, it will continue to run!

Here is a simple example where we want to continue to generate values until we obtain a positive x.

```
set.seed(17)
x<-rnorm(1, mean=0, sd=1)
x
```

```
## [1] -1.015009
```

```
while(x<0)
{
  x<-rnorm(1, mean=0, sd=1)
  print(x)
}
```

```
## [1] -0.07963674
## [1] -0.232987
## [1] -0.8172679
## [1] 0.7720908
```

Here, we first generate a negative (-1.01). Then, it runs the loop 4 times! The first three times each generate a negative, before finally we obtain a positive. Be cautious using while-loops, as if the condition is never met, it will run forever!

Assignment Week 2

This assignment can be completed in the Math and Stats Computer lab (ADEL 222). If you choose to work on a personal computer, be prepared that R Markdown may not compile PDFs due to issues with Latex interfaces. Although work can be done on a personal computer, if issues with creating a PDF are encountered, you should use ADEL 222 to finalize your assignment.

The goal of the assignment is to introduce you to the aspects of the tidyverse and base R, but to not overwhelm you in a single week. If you are interested in R, and want to learn more, I suggest the STA 570L, or consulting some of the many texts that I suggested in the first presentation. Some of what will need to be done, you will have to look up. Use Google, the online textbook, or ask questions if necessary! The best way to learn is to try!

The assignment must be turned in as a PDF created through R markdown.

You will need to load (and possibly install) the packages: Lock5Data, ggplot2, dplyr, tidyr

1. Load the data set ColaCalcium from the Lock5Data package. This data set provides calcium excretion based on drinking diet cola or water.
2. Using dplyr commands, calculate the mean, standard deviation, and sample size for the Diet cola and water samples. The command to obtain sample size is simply "Sample.Size=n()". Be sure to name all the variables something simple. Display the summary statistics to screen. (Hint: You will need to use the following commands in some way: *group_by*, *summarise*, *mean*, *sd*, *n*. Don't forget about the pipe commands %>%. Use my examples to help!)
3. Using the ggplot2, create side-by-side boxplots for the amount of Calcium excreted.
4. Load in the data frame provided below:

```
grade.book <- rbind(
  data.frame(name='Alison', HW.1=8, HW.2=5, HW.3=8, HW.4=4),
  data.frame(name='Brandon', HW.1=5, HW.2=3, HW.3=6, HW.4=9),
  data.frame(name='Charles', HW.1=9, HW.2=7, HW.3=9, HW.4=10))
grade.book
```

5. Using the tidyr command *gather*, create a new data frame called grade.book.long with three columns. The columns should be named: name, Homework, Score. Be sure to display the result.
6. Right a for loop that calculates the value of sin() from -2π to 2π every $\pi/8$. Use the *print()* command to display answers to screen for each iteration.

Be sure to put each question as text above the code blocks. I want to see both the question and the answers!

Some of these are tough! Give your best effort and see what you can learn!

Print the .PDF and submit in class next week! I encourage you to stop by my lab hours T/Th 2:20 - 3:10 PM if you have questions about R!