Session 3

Nick Hathaway; nicholas.hathaway@umassmed.edu

Contents

Set up

Create a directory for today's session and change your workding to this directory. Next download these three datasets and put them into this directory, World Temp Max, World Temp Min, World Temp Avg. Once you have downloadt the datasets, they are compressed files they need to be opened, the decompression, downloading, and creating of the directory don't need to be done in R.

Once you have downloaded the files, take a glance at data.txt and site_detail.txt and notice how there are a lot of lines of information and not data at the top of the files that all start with `%`. Some data files may come like this where you need to ignore certain lines that contain comments, luckily our friend `readr` package allows to say that we have comment lines. You can also see that we don't have colnames

Reading In Data and manipulating

```
require(readr)
require(dplyr)
require(tidyr)
#read in average temperature for the world in the 90's
tavg = read_tsv("tavg_worldTemp/data.txt", col_names = F, comment = "%")
#read in meta data
tavg_meta = read_tsv("tavg_worldTemp/site_detail.txt", col_names = F, comment = "%")
```

`col_names = F` will make it so read_tsv will not assume the first one is a column name line.`comment = "%"` will make read_tsv ignore all lines that start with `%`.

Now that we have read in the data we will need to set the column names of the data.frame so that we can make sense of the data. Looking at the comment lines we can see that the columns are `"Station`

ID", "Series Number", "Date", "Temperature (C)", "Uncertainty (C)","Observations","Time of Observation" .

```
print(tavg)
```

```
# A tibble: 206,384 x 7
      X1    X2       X3    X4    X5    X6    X7
   <int> <int>    <dbl> <dbl> <dbl> <int> <int>
1   1155     1 1991.042  10.8  0.05   -99   -99
2   1155     1 1991.125   8.4  0.05   -99   -99
3   1155     1 1991.208  13.6  0.05   -99   -99
4   1155     1 1991.292  13.0  0.05   -99   -99
5   1155     1 1991.375  14.5  0.05   -99   -99
6   1155     1 1991.458  19.2  0.05   -99   -99
7   1155     1 1991.542  24.1  0.05   -99   -99
8   1155     1 1991.625  24.8  0.05   -99   -99
9   1155     1 1991.708  22.7  0.05   -99   -99
10  1155     1 1991.792  16.3  0.05   -99   -99
# ... with 206,374 more rows
```

```
colnames(tavg) = c("Station ID", "Series Number", "Date", "Temperature (C)", "Uncertainty (C)","Observa
print(tavg)
```

```
# A tibble: 206,384 x 7
   Station ID Series Number     Date Temperature (C) Uncertainty (C) Observations
        <int>         <int>    <dbl>           <dbl>           <dbl>        <int>
1        1155             1 1991.042            10.8            0.05          -99
2        1155             1 1991.125             8.4            0.05          -99
3        1155             1 1991.208            13.6            0.05          -99
4        1155             1 1991.292            13.0            0.05          -99
5        1155             1 1991.375            14.5            0.05          -99
6        1155             1 1991.458            19.2            0.05          -99
7        1155             1 1991.542            24.1            0.05          -99
8        1155             1 1991.625            24.8            0.05          -99
9        1155             1 1991.708            22.7            0.05          -99
10       1155             1 1991.792            16.3            0.05          -99
# ... with 206,374 more rows, and 1 more variables: Time of Observation <int>
```

`colnames()` can both be used to simply see the what the colnames of data.frame are but can also be used to set the column names. Now R doesn't like it when there are spaces in the column names, so we will use gsub to replace all spaces with _ and use colnames again to set the colnames.

```
colnames(tavg) = gsub(" ", "_", colnames(tavg))
print(tavg)
```

```
# A tibble: 206,384 x 7
   Station_ID Series_Number     Date Temperature_(C) Uncertainty_(C) Observations
        <int>         <int>    <dbl>           <dbl>           <dbl>        <int>
1        1155             1 1991.042            10.8            0.05          -99
2        1155             1 1991.125             8.4            0.05          -99
3        1155             1 1991.208            13.6            0.05          -99
4        1155             1 1991.292            13.0            0.05          -99
```

```
5        1155             1 1991.375            14.5            0.05          -99
6        1155             1 1991.458            19.2            0.05          -99
7        1155             1 1991.542            24.1            0.05          -99
8        1155             1 1991.625            24.8            0.05          -99
9        1155             1 1991.708            22.7            0.05          -99
10       1155             1 1991.792            16.3            0.05          -99
# ... with 206,374 more rows, and 1 more variables: Time_of_Observation <int>
```

gsub takes three arguments, 1) the pattern to replace, 2) the replacement, 3) the vector to perform the replacement on

Now we don't need all the columns and it looks like some are place holders and don't contain real data. To make sure this is true, i like to use the `table` function. It take a vector and counts up all the elements in it and is very useful to get a quick count of elements.

```
print(table(tavg$Series_Number))
```

```
     1
206384
```

```
print(table(tavg$Observations))
```

```
   -99
206384
```

We can see that these columns don't contain anything useful so lets select only the colunms we want

```
tavg_sel = select(tavg, one_of("Station_ID", "Date", "Temperature_(C)"))

print(tavg_sel)
```

```
# A tibble: 206,384 x 3
   Station_ID     Date Temperature_(C)
        <int>    <dbl>           <dbl>
1        1155 1991.042            10.8
2        1155 1991.125             8.4
3        1155 1991.208            13.6
4        1155 1991.292            13.0
5        1155 1991.375            14.5
6        1155 1991.458            19.2
7        1155 1991.542            24.1
8        1155 1991.625            24.8
9        1155 1991.708            22.7
10       1155 1991.792            16.3
# ... with 206,374 more rows
```

Here we take advantage one of dplyr special functions `one_of` which take the columns that one of the ones given here.

Now we also want to rename the meta data columns names as well.

```
metaCols = "Station ID, Station Name, Latitude, Longitude, Elevation (m), Lat. Uncertainty, Long. Uncer

colnames(tavg_meta) = gsub(" ", "_",unlist(strsplit(metaCols, ", ")))
```

Here I simply copied the line that contained the columns names from the site_detail.txt file. I then take advantage of the R function `strsplit()` which allows you to split a chracter string on a certain delimiter and return a list with the first item being a vector of the words split which is why we have tu unlist() the result. We then use the same gsub command above to replace all spaces with _.

Also we are only interested in certain meta data so lets select for those columns, like above.

```
tavg_meta = select(tavg_meta, one_of("Station_ID", "Station_Name", "Latitude", "Longitude", "Country",
```

Next, since we are going to be working the average, min, and max temperature we are going to want to edit the column names of the data. We can do this again with colnames() function but instead of changing all the columns names we can actually now just change just the one we want.

```
print(tavg_sel)
```

```
# A tibble: 206,384 x 3
   Station_ID     Date Temperature_(C)
        <int>    <dbl>           <dbl>
1        1155 1991.042            10.8
2        1155 1991.125             8.4
3        1155 1991.208            13.6
4        1155 1991.292            13.0
5        1155 1991.375            14.5
6        1155 1991.458            19.2
7        1155 1991.542            24.1
8        1155 1991.625            24.8
9        1155 1991.708            22.7
10       1155 1991.792            16.3
# ... with 206,374 more rows
```

```
colnames(tavg_sel)[3] = "Temp_Avg"
print(tavg_sel)
```

```
# A tibble: 206,384 x 3
   Station_ID     Date Temp_Avg
        <int>    <dbl>    <dbl>
1        1155 1991.042     10.8
2        1155 1991.125      8.4
3        1155 1991.208     13.6
4        1155 1991.292     13.0
5        1155 1991.375     14.5
6        1155 1991.458     19.2
7        1155 1991.542     24.1
8        1155 1991.625     24.8
9        1155 1991.708     22.7
10       1155 1991.792     16.3
# ... with 206,374 more rows
```

Here we change the third column name to Temp_Avg

Part 1 Exercises

1. Read in the data and meta data of the minimum and maximum temperatures as well and modify/select the columns like we did for the average temperature.

Combining Datasets

Now that we have read in the average, min, and max temperatures we want to combine the datasets into one data.frame to make furth work with the data easier. We can accomplish this with `dplyr`'s left_join. Which can join data based on matching values in one or more columns

Since all three datasets share the columns `"Station_ID", "Date"` we'll combine the data using those.

```
#first join two of the datasets
temps = left_join(tmax_sel, tmin_sel, by = c("Station_ID", "Date"))
#then join in the last one
temps = left_join(temps, tavg_sel, by = c("Station_ID", "Date"))
print(temps)
```

```
# A tibble: 210,155 x 5
   Station_ID     Date Temp_Max Temp_Min Temp_Avg
        <int>    <dbl>    <dbl>    <dbl>    <dbl>
1        1155 1991.042     13.2      8.4     10.8
2        1155 1991.125     11.2      5.5      8.4
3        1155 1991.208     16.2     10.9     13.6
4        1155 1991.292     16.0      9.9     13.0
5        1155 1991.375     17.0     12.0     14.5
6        1155 1991.458     22.0     16.5     19.2
7        1155 1991.542     27.2     20.9     24.1
8        1155 1991.625     27.9     21.8     24.8
9        1155 1991.708     25.5     19.8     22.7
10       1155 1991.792     19.2     13.3     16.3
# ... with 210,145 more rows
```

We now have one dataset for the three and since we change the temperature column names we have three distinct columns, but now we want to have more information than just "Station_ID" which doesn't really tell us too much, so we'll now join in the data from one of the meta datasets that we read in.

```
temps = left_join(temps, tmax_meta, by = "Station_ID")
print(temps)
```

```
# A tibble: 210,155 x 10
   Station_ID     Date Temp_Max Temp_Min Temp_Avg                Station_Name Latitude Longitude
        <int>    <dbl>    <dbl>    <dbl>    <dbl>                       <chr>    <dbl>     <dbl>
1        1155 1991.042     13.2      8.4     10.8 Missing Station ID - 99990     -999      -999
2        1155 1991.125     11.2      5.5      8.4 Missing Station ID - 99990     -999      -999
3        1155 1991.208     16.2     10.9     13.6 Missing Station ID - 99990     -999      -999
4        1155 1991.292     16.0      9.9     13.0 Missing Station ID - 99990     -999      -999
5        1155 1991.375     17.0     12.0     14.5 Missing Station ID - 99990     -999      -999
6        1155 1991.458     22.0     16.5     19.2 Missing Station ID - 99990     -999      -999
7        1155 1991.542     27.2     20.9     24.1 Missing Station ID - 99990     -999      -999
```

```
8        1155 1991.625     27.9     21.8     24.8 Missing Station ID - 99990     -999      -999
9        1155 1991.708     25.5     19.8     22.7 Missing Station ID - 99990     -999      -999
10       1155 1991.792     19.2     13.3     16.3 Missing Station ID - 99990     -999      -999
# ... with 210,145 more rows, and 2 more variables: Country <chr>, State_/_Province_Code <chr>
```

Now we have all the meta data that corresponds to "Station_ID" from the meta data file in our datafile.

However our data has a date column that we want to query so we can get the actual month names rather than a fraction. To does we can use the function seperate but first we need to change the type of data of date to character so we can do certain operations on it which we use the `as.character` function.

```
temps$Date = as.character(temps$Date)
print(temps)
```

```
# A tibble: 210,155 x 10
   Station_ID     Date Temp_Max Temp_Min Temp_Avg               Station_Name Latitude Longitude
        <int>    <chr>    <dbl>    <dbl>    <dbl>                      <chr>    <dbl>     <dbl>
1        1155 1991.042     13.2      8.4     10.8 Missing Station ID - 99990     -999      -999
2        1155 1991.125     11.2      5.5      8.4 Missing Station ID - 99990     -999      -999
3        1155 1991.208     16.2     10.9     13.6 Missing Station ID - 99990     -999      -999
4        1155 1991.292     16.0      9.9     13.0 Missing Station ID - 99990     -999      -999
5        1155 1991.375     17.0     12.0     14.5 Missing Station ID - 99990     -999      -999
6        1155 1991.458     22.0     16.5     19.2 Missing Station ID - 99990     -999      -999
7        1155 1991.542     27.2     20.9     24.1 Missing Station ID - 99990     -999      -999
8        1155 1991.625     27.9     21.8     24.8 Missing Station ID - 99990     -999      -999
9        1155 1991.708     25.5     19.8     22.7 Missing Station ID - 99990     -999      -999
10       1155 1991.792     19.2     13.3     16.3 Missing Station ID - 99990     -999      -999
# ... with 210,145 more rows, and 2 more variables: Country <chr>, State_/_Province_Code <chr>
```

Now we can seperate out by the "." to get year the month data and then convert the numbers back to actual numbers.

```
temps = separate(temps, Date, c("Year", "Month"), sep = c("\\."))

temps$Year = as.numeric(temps$Year)
temps$Month = as.numeric(temps$Month)

print(temps)
```

```
# A tibble: 210,155 x 11
   Station_ID  Year Month Temp_Max Temp_Min Temp_Avg               Station_Name Latitude Longitude
*       <int> <dbl> <dbl>    <dbl>    <dbl>    <dbl>                      <chr>    <dbl>     <dbl>
1        1155  1991    42     13.2      8.4     10.8 Missing Station ID - 99990     -999      -999
2        1155  1991   125     11.2      5.5      8.4 Missing Station ID - 99990     -999      -999
3        1155  1991   208     16.2     10.9     13.6 Missing Station ID - 99990     -999      -999
4        1155  1991   292     16.0      9.9     13.0 Missing Station ID - 99990     -999      -999
5        1155  1991   375     17.0     12.0     14.5 Missing Station ID - 99990     -999      -999
6        1155  1991   458     22.0     16.5     19.2 Missing Station ID - 99990     -999      -999
7        1155  1991   542     27.2     20.9     24.1 Missing Station ID - 99990     -999      -999
8        1155  1991   625     27.9     21.8     24.8 Missing Station ID - 99990     -999      -999
9        1155  1991   708     25.5     19.8     22.7 Missing Station ID - 99990     -999      -999
10       1155  1991   792     19.2     13.3     16.3 Missing Station ID - 99990     -999      -999
# ... with 210,145 more rows, and 2 more variables: Country <chr>, State_/_Province_Code <chr>
```

But now our month column is messed up so now we have to change it so we can actually make sense of the numbers. We will use dplyr's function `mutate` which allows us to create a new column by doing calculations based on the other columns. Here we used the fact that in the data file data.txt we now that they encoded the month data as the following.

```
% For example, in monthly data:
%
%   January 2005 = 2005 + (1 - 0.5) / 12 = 2005.042
%   June 2008 = 2008 + (6 - 0.5) / 12 = 2008.458
```

```
temps = mutate(temps, Month = round((Month/1000) * 12 + 0.5 ))
```

And then we can take advantage of the fact that R has a vector already of the months called `month.name` and that we can index into to get the month names

```
print(month.name)
```

```
 [1] "January"   "February"  "March"     "April"     "May"       "June"      "July"      "August"
 [9] "September" "October"   "November"  "December"
```

```
temps = mutate(temps, MonthName = month.name[Month])
print(temps["MonthName"])
```

```
# A tibble: 210,155 x 1
   MonthName
       <chr>
1    January
2   February
3      March
4      April
5        May
6       June
7       July
8     August
9  September
10   October
# ... with 210,145 more rows
```

We have now manipulated our original data into something usage that we now get the data we want out of it. For example if we want to get all the temperatures for the united states we can do

```
temps_usa = filter(temps, Country == "United States")

print(temps_usa)
```

```
# A tibble: 33,174 x 12
   Station_ID  Year Month Temp_Max Temp_Min Temp_Avg Station_Name Latitude Longitude       Country
        <int> <dbl> <dbl>    <dbl>    <dbl>    <dbl>        <chr>    <dbl>     <dbl>         <chr>
1       25605  1991     1     24.6     17.7     21.1   KAILUA/446     20.9 -156.2167 United States
2       25605  1991     2     25.8     18.2     22.0   KAILUA/446     20.9 -156.2167 United States
3       25605  1991     3     26.4     18.6     22.5   KAILUA/446     20.9 -156.2167 United States
```

```
4       25605  1991     4     26.8     19.7     23.3   KAILUA/446     20.9 -156.2167 United States
5       25605  1991     5     27.8     18.8     23.3   KAILUA/446     20.9 -156.2167 United States
6       25605  1992     1     24.9     15.3     20.1   KAILUA/446     20.9 -156.2167 United States
7       25605  1992     2     24.5     16.8     20.7   KAILUA/446     20.9 -156.2167 United States
8       25605  1992     3     26.1     16.4     21.3   KAILUA/446     20.9 -156.2167 United States
9       25605  1992     4     26.9     19.1     23.1   KAILUA/446     20.9 -156.2167 United States
10      25605  1992     5     27.1     19.8     23.5   KAILUA/446     20.9 -156.2167 United States
# ... with 33,164 more rows, and 2 more variables: State_/_Province_Code <chr>, MonthName <chr>
```

Or if we want get the temps from a certain year and from only some cities. Like below we get only the data from the Year 2000 and from the cites "SAN FRANCISCO/INTERNATIO", "BOSTON/LOGAN INT'L ARPT".

```
temps_usa_filtered = filter(temps_usa, Year ==2000, Station_Name %in% c("SAN FRANCISCO/INTERNATIO", "BOS
print(temps_usa_filtered)
```

```
# A tibble: 24 x 12
   Station_ID  Year Month Temp_Max Temp_Min Temp_Avg             Station_Name Latitude Longitude
        <int> <dbl> <dbl>    <dbl>    <dbl>    <dbl>                    <chr>    <dbl>     <dbl>
1       87406  2000     1     14.3      8.3     11.3 SAN FRANCISCO/INTERNATIO 37.61667 -122.3833
2       87406  2000     2     14.8      9.1     11.9 SAN FRANCISCO/INTERNATIO 37.61667 -122.3833
3       87406  2000     3     16.4      8.4     12.4 SAN FRANCISCO/INTERNATIO 37.61667 -122.3833
4       87406  2000     4     18.7     10.4     14.6 SAN FRANCISCO/INTERNATIO 37.61667 -122.3833
5       87406  2000     5     20.0     11.3     15.7 SAN FRANCISCO/INTERNATIO 37.61667 -122.3833
6       87406  2000     6     21.7     12.6     17.2 SAN FRANCISCO/INTERNATIO 37.61667 -122.3833
7       87406  2000     7     20.1     12.5     16.3 SAN FRANCISCO/INTERNATIO 37.61667 -122.3833
8       87406  2000     8     22.2     12.9     17.6 SAN FRANCISCO/INTERNATIO 37.61667 -122.3833
9       87406  2000     9     24.1     13.9     19.0 SAN FRANCISCO/INTERNATIO 37.61667 -122.3833
10      87406  2000    10     19.4     11.7     15.6 SAN FRANCISCO/INTERNATIO 37.61667 -122.3833
# ... with 14 more rows, and 3 more variables: Country <chr>, State_/_Province_Code <chr>,
#   MonthName <chr>
```

Part 2 Exercises

1. Combine the datasets like we did above and convert the month vector over to the correct set up
2. Filter down the data to your favorite country or several of your favorite cities.

3. If you are feeling adventurous, try convert temperature from C to F.

Plotting

For plotting we will be using `ggplot2` which has extensive plotting features. The interface of it takes a little bit of time to get use to but once you do it's pretty easy to manipulate. To see all the things that ggplot2 can do you can visit their site which has lots of examples, http://docs.ggplot2.org/current/

I will be showing just a few examples. First lets try a simple line plot of two cities of temperatures over a single year

```
temps_usa_filtered = filter(temps_usa, Year ==2000, Station_Name %in% c("SAN FRANCISCO/INTERNATIO", "BOS
```

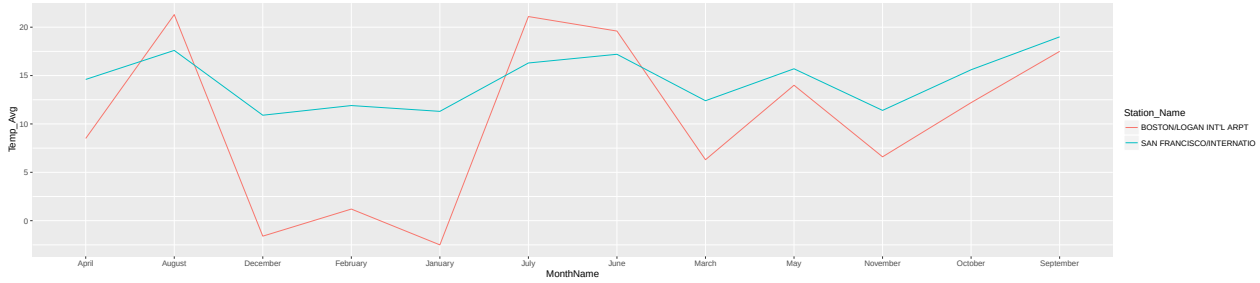
The way ggplot2 works is by creating objects that contain all the information that it needs to plot and will only plot once you call `print()` on this plot object.

```
library(ggplot2)
plotObj = ggplot(temps_usa_filtered, aes(x = MonthName, y = Temp_Avg, group = Station_Name, color = Sta
plotObj = plotObj + geom_line()

#lets look at what the class is for plotObj
print(class(plotObj))
```

```
[1] "gg"     "ggplot"
```

```
#now lets call print(plotObj) to generate the plot
print(plotObj)
```

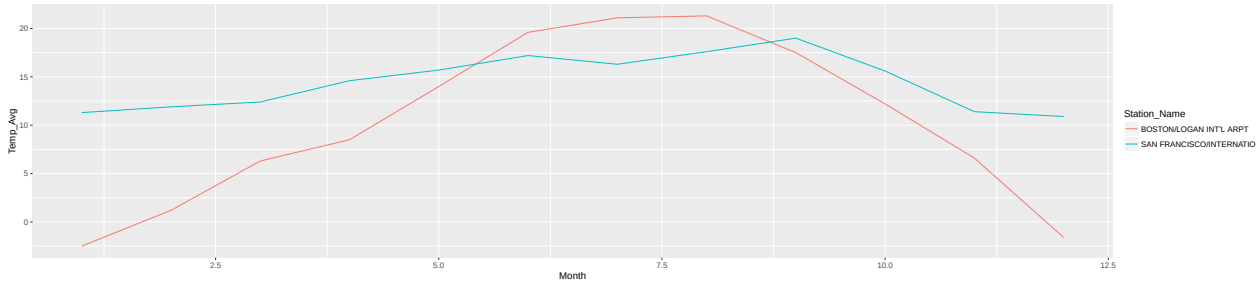


Now this call might look intimidating and weird. So lets break it down a little bit. The first call `ggplot` will generate our base object. We need to give this our data.frame to work from, which is our first argument to ggplot. Next we want to describe to ggplot how we want the plot to look so we want to describe it's aesthetics, which you do by using the function `aes()`. In this function you tell it what will be on the x axis by giving the column name (here MonthName) and what will be on the y axis. Next we want to have separate plotting for our Stations and we want to color them by the station names as well. Next we need to add to our plot object what type of plot we want, which for ggplot that is done with functions that all start with geom_. There are many options but most will have the same base `ggplot()` call. You can visit the site to see all the exaples and we will look at one more today.

But first can any one see that there is something wrong with out plot?

The x axis was ordered by alphabetical order but unfortunately that means that our months come in the wrong order. We could solve this by using the column that had the month numbers instead.

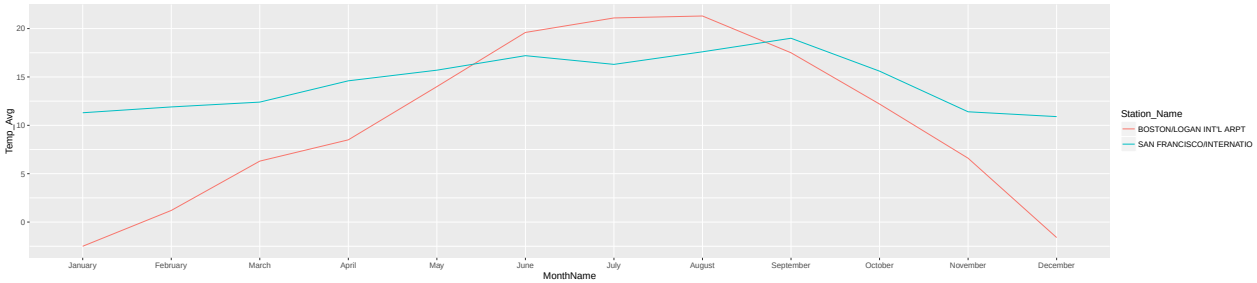
```
library(ggplot2)
plotObj = ggplot(temps_usa_filtered, aes(x = Month, y = Temp_Avg, group = Station_Name, color = Station_
plotObj = plotObj + geom_line()
#now lets call print(plotObj) to generate the plot
print(plotObj)
```



Oh but now the x axis labeling is kind of funny, so it would be best to keep using the month dates. Here we

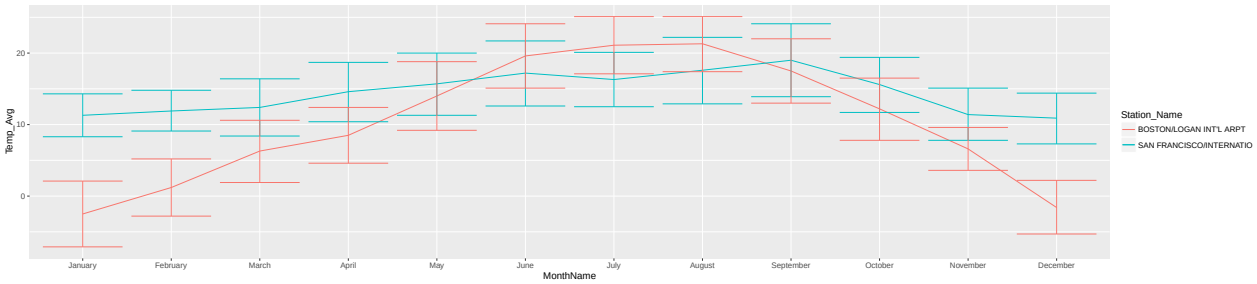
can take advantage of converting the months into Factors rather than the characters they are now. We can then tell R what order they should be ordered in, like below. (here we take advantage that `month.name` orders the months correctly)

```
temps_usa_filtered$MonthName = factor(temps_usa_filtered$MonthName, levels = month.name)
library(ggplot2)
plotObj = ggplot(temps_usa_filtered, aes(x = MonthName, y = Temp_Avg, group = Station_Name, color = Sta
plotObj = plotObj + geom_line()
#now lets call print(plotObj) to generate the plot
print(plotObj)
```



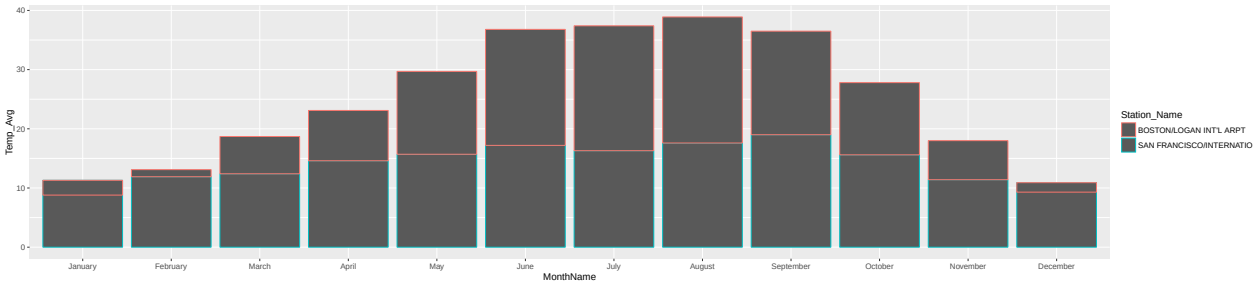
Ah there we go, now it would be interesting to see the the mins and max as well so lets add another layer to our plot object

```
library(ggplot2)
plotObj = ggplot(temps_usa_filtered, aes(x = MonthName, y = Temp_Avg, group = Station_Name, color = Sta
plotObj = plotObj + geom_line()
plotObj = plotObj + geom_errorbar(aes(ymin = Temp_Min, ymax = Temp_Max))
#now lets call print(plotObj) to generate the plot
print(plotObj)
```



Now say we wanted to view the data not as a line plot but as a bar plot instead.

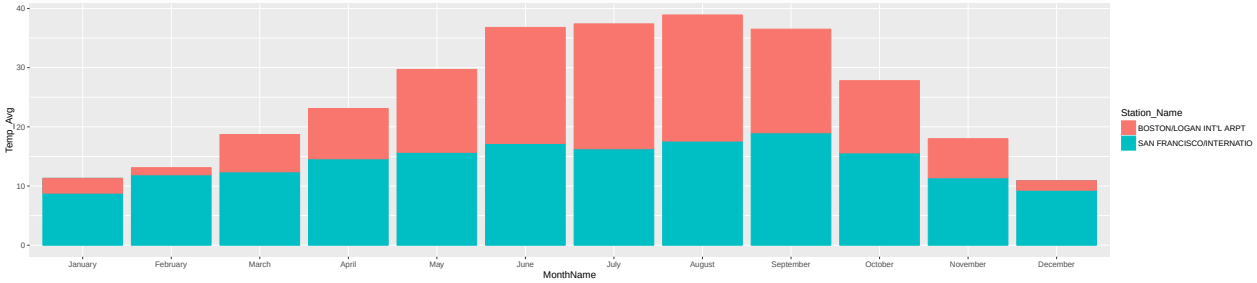
```
library(ggplot2)
plotObj = ggplot(temps_usa_filtered, aes(x = MonthName, y = Temp_Avg, group = Station_Name, color = Sta
plotObj = plotObj + geom_bar(stat = "identity")
#now lets call print(plotObj) to generate the plot
print(plotObj)
```



Here we had to tell geom_bar that we are supplying y values, by default it will do a histogram of counts in what we gave for `x`.

Also want to fill our bar rather than just color.

```
library(ggplot2)
plotObj = ggplot(temps_usa_filtered, aes(x = MonthName, y = Temp_Avg, group = Station_Name, fill = Stati
plotObj = plotObj + geom_bar(stat = "identity")
#now lets call print(plotObj) to generate the plot
print(plotObj)
```



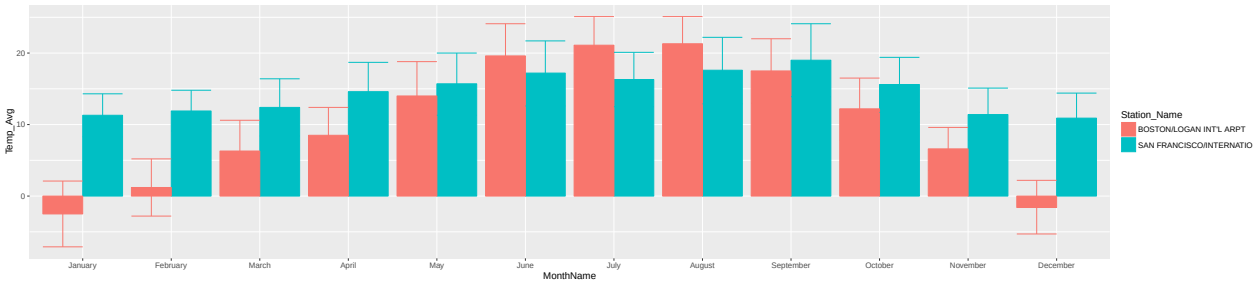
But having our bar stacked isn't very helpful so lets tell geom_bar to put them next to each other, which this isn't done very straight forwardly but adding position = "dodge" will do it.

```
library(ggplot2)
plotObj = ggplot(temps_usa_filtered, aes(x = MonthName, y = Temp_Avg, group = Station_Name, fill = Stati
plotObj = plotObj + geom_bar(stat = "identity",  position = "dodge")
#now lets call print(plotObj) to generate the plot
print(plotObj)
```

And lets add back in the error bars which is only done properly if we dodge those as well.

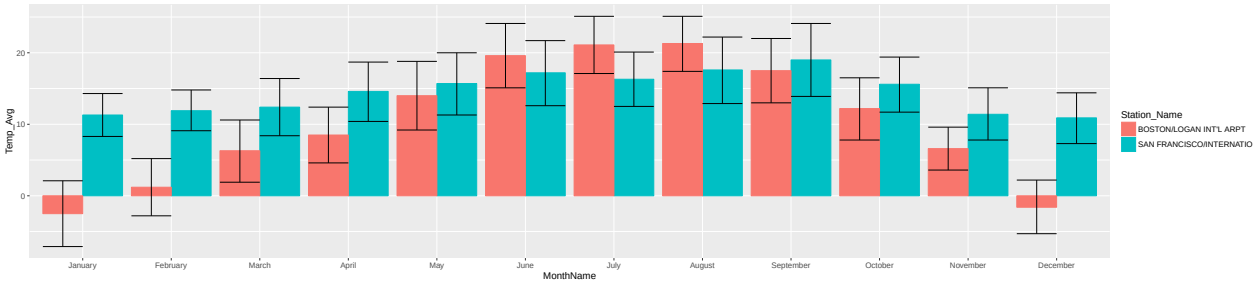
```
library(ggplot2)
plotObj = ggplot(temps_usa_filtered, aes(x = MonthName, y = Temp_Avg, group = Station_Name, fill = Stati
plotObj = plotObj + geom_bar(stat = "identity",  position = "dodge")
plotObj = plotObj + geom_errorbar(aes(ymin = Temp_Min, ymax = Temp_Max), position = "dodge")
```

```
#now lets call print(plotObj) to generate the plot
print(plotObj)
```



Lets change their colors to black though

```
library(ggplot2)
plotObj = ggplot(temps_usa_filtered, aes(x = MonthName, y = Temp_Avg, group = Station_Name, fill = Stati
plotObj = plotObj + geom_bar(stat = "identity",  position = "dodge")
plotObj = plotObj + geom_errorbar(aes(ymin = Temp_Min, ymax = Temp_Max), position = "dodge", color = "bl
#now lets call print(plotObj) to generate the plot
print(plotObj)
```



Part 3 Exercises

1. Try creating some lines and bar plots comparing some cities or countries of your choosing.