03-02 Data Frame Manipulation

1 - Purpose

- Load data frame from a CSV and save data frame to a CSV
- Reshape a data frame
- Perform conversion and formulas on multiple column of a data frame
- Rename column headers

2 - Concepts

3 - A script for this lesson

The script that contains all of the code for this lesson can be found here. This lesson, and all subsequent lessons, will not contain embedded scripts that you copy/paste. Instead the lessons will have the full, downloadable script linked at the beginning and the lesson will go through the linked script in order.

If the NOAA/NCDC website is down, you will need to download this rdata file, which contains the weather data for this lesson. Save this file to your ***data*** folder in the ***R Root*** directory. The script file for this lesson has instructions on how to adapt the script for the rdata file.

3.1 - Commenting/ uncommenting multiple lines

Since the linked script above follows the lesson in order, you probably want to comment out the whole script and remove the comments as you go through the lesson so that you can see the progression of the lesson. In RStudio, you can comment lines by (*Fig 1*):
1) highlighting the lines you want to comment
2) clicking ***Code -> Comment/Uncomment Lines*** or ***Control-Shift-C*** (***Command-Shift-C*** on a MAC).

Just *make sure you don't comment the curly bracket (**}**)* at the end of your code.

You can uncomment lines using the same technique (*Fig 1*) or by deleting the comments symbols (**#**) at the beginning of the line.

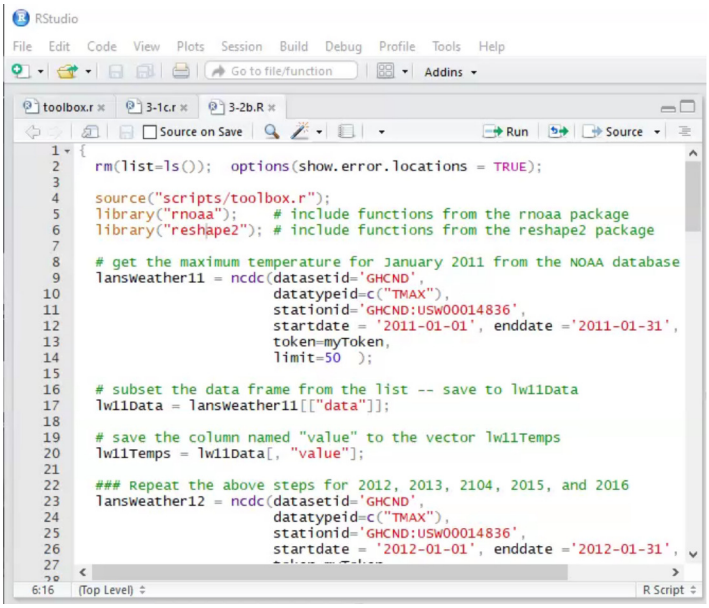


Fig 1: (Video) Commenting and uncommenting blocks of code in RStudio.

4 - Getting the weather data

Like last lesson, we are going to link to ***toolbox.r*** to get the token and use the library ***rnoaa*** to connect to the NOAA/NCDC database. We will also use the library ***reshape2*** to manipulate the data frame that comes back from the NOAA/NCDC database.

```
source("scripts/toolbox.r");
library(rnoaa);
library(reshape2);
```

Like last lesson we will call the NOAA/NCDC database using ***ncdc()*** and get weather data from the Lansing station for January 2016.

```
lansingWeather = ncdc(datasetid="GHCND",
          datatypeid=c("TMAX", "TAVG", "TMIN", "PRCP",
                       "SNOW", "AWND", "WDF2", "WSF2",
                       "WT01", "WT09"),
          stationid="GHCND:USW00014836",
          startdate = "2016-01-01", enddate ="2016-01-31",
          token=myToken,
          limit=400 );
```

Extension: The four-digit data type ids

Next, we are going to extract the data frame, ***data***, from the list we got from NOAA/NCDC, ***lansingWeather***.

```
lansingWeatherDF = lansingWeather[["data"]];
```

5 - Reshaping a data frame

The data frame ***lansingWeatherDF*** is not in a very readable format. This will often happen when you get data from a big database. The database is just throwing data at you -- it is your job to format the data in a useful way.

In the data frame, every weather measurement from every day has its own row. So, there are 10 rows with the date **01-02** and each row has a different weather type (**TMAX, TMIN, TAVG**, ...).

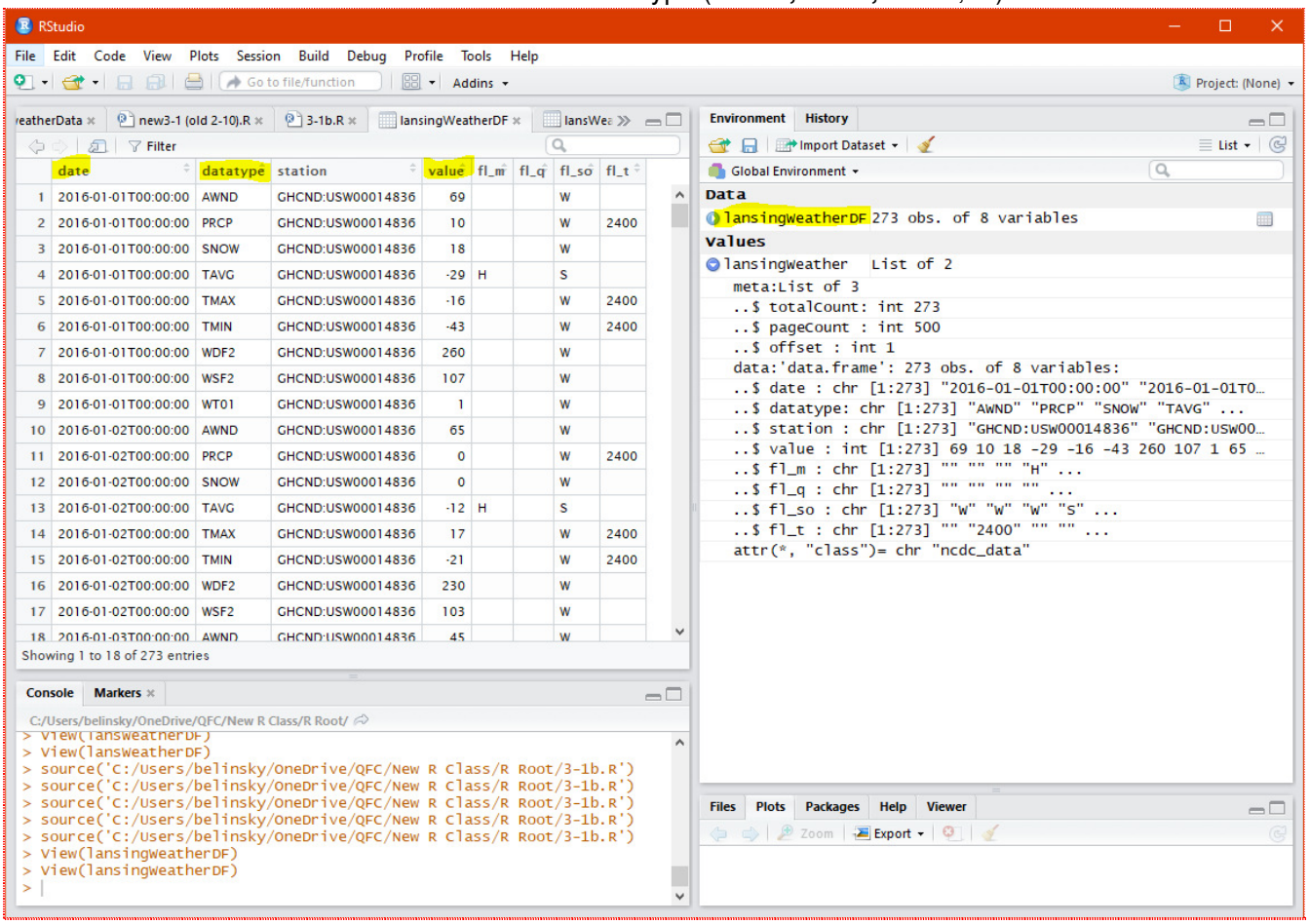


Fig 2: Data frame that needs serious formatting.

5.1 - The data frame we want

We would like the data to look like this (partial table, values are from ***lansingWeatherDF***):

	minTemp	avgTemp	maxTemp	...	Precip
Jan1	24	26	29	...	0.04
Jan2	28	32	35	...	0
Jan3	25	29	33	...	T
...	...	...	...	...	...
Jan31	33	44	49	...	0

In other words we want:

1) unique *dates* on each row
2) unique *weather types* on each column
3) values for *weather types* for each *date* populating the cells

The first thing to notice is that there are really only three columns in the data frame (*Fig 2*) that are useful to us: **date**, **datatype**, and **value**.

So, what we want is to:
1) make each unique ***date*** (31 in all) a row
2) make each unique ***datatype*** (10 in all) a column
3) populated the table cells with the information in the ***value*** column.

5.2 - Compressing and reshaping data

This is where we use the **reshape2** package.

Information about the **reshape2** package can be found here. We are going to use the ***dcast()*** function on pages 2 and 3. There are many parameters to ***dcast()*** but we are only concerned with three of the parameters:

1) ***data***: the data frame we are reshaping, which is ***lansingWeatherDF***
2) ***formula***: defines the ***x-axis*** and ***y-axis*** of the reshaped data frame
   - formula takes the form ***y-axis ~ x-axis.***
   - In this case we want ***dataype*** on the ***y-axis*** and ***dates*** on the ***x-axis***
   - So the formula is: ***date ~ datatype***
3) ***value.var***: determines which column populates the rest of the reshaped data frame -- the column that holds the values in ***lansingWeatherDF*** is named ***value***
   - ***value*** is in quotes in the formula because ***value*** is a column header, *not a variable*

```
1  lansingWeatherRS = dcast(data = lansingWeatherDF,
2           formula = date ~ datatype,
3           value.var = "value");
```

The reshaped data frame gets saved to the variable ***lansingWeatherRS*** and looks like this...

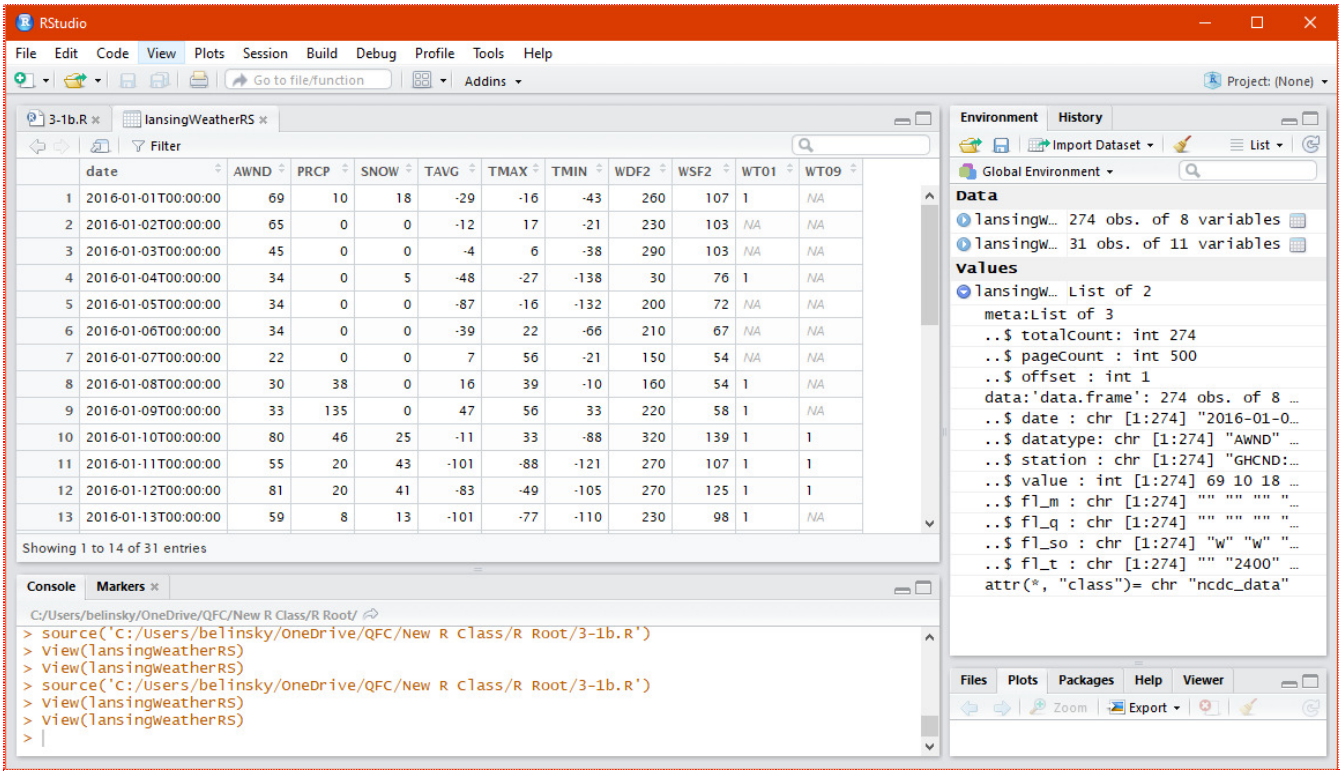


Fig 3: Lansing weather data frame after reshaping.

Extension: functions with the same name

6 - Manipulating the data frame

Now that we have the data frame shaped the way we want, we can make other manipulations to the data.

6.1 - Reordering the columns

We can reorder the columns by following these steps:
1) get the columns from the data frame in the order we want
2) save the columns back to the same data frame variable, which effectively overwrites the variable using the new column order

This is conceptually very similar to saying:

```
1  count = count + 1;
```

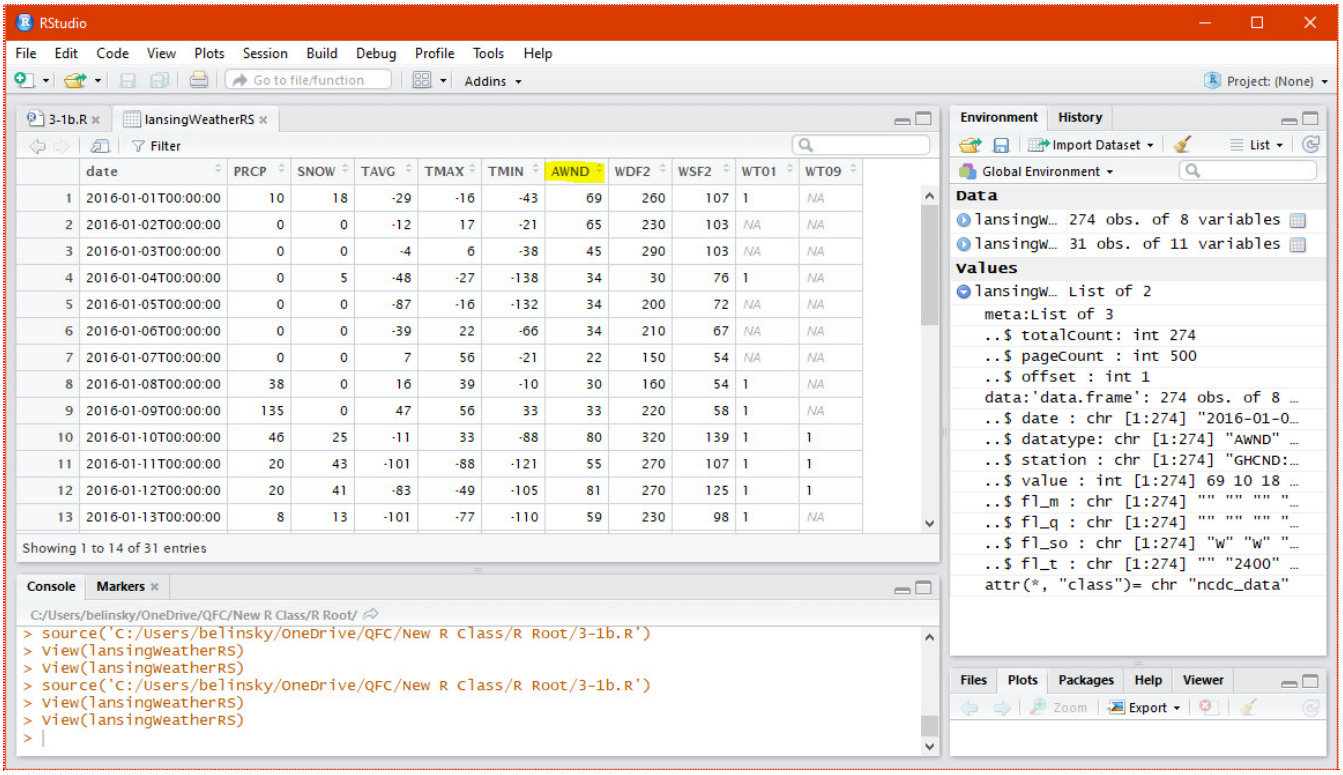
In R, this means add one to the value of ***count*** and then save the answer to ***count.***

To reorder the columns we add the code:

```
1  lansingWeatherRS = lansingWeatherRS[,c(1,3,4,5,6,7,2,8,9,10,11)];
```

The code above says we will:
1) get the columns from ***lansingWeatherRS*** in this order: 1,3,4,5,6,7,2,8,9,10,11 (right side)
2) save this column order back to ***lansingWeatherRS*** (left side).

Effectively, we moved column 2, which was ***AWND,*** to column 7 and shifted the columns in between.



	date	PRCP	SNOW	TAVG	TMAX	TMIN	AWND	WDF2	WSF2	WT01	WT09
1	2016-01-01T00:00:00	10	18	-29	-16	-43	69	260	107	1	NA
2	2016-01-02T00:00:00	0	0	-12	17	-21	65	230	103	NA	NA
3	2016-01-03T00:00:00	0	0	-4	6	-38	45	290	103	NA	NA
4	2016-01-04T00:00:00	0	5	-48	-27	-138	34	30	76	1	NA
5	2016-01-05T00:00:00	0	0	-87	-16	-132	34	200	72	NA	NA
6	2016-01-06T00:00:00	0	0	-39	22	-66	34	210	67	NA	NA
7	2016-01-07T00:00:00	0	0	7	56	-21	22	150	54	NA	NA
8	2016-01-08T00:00:00	38	0	16	39	-10	30	160	54	1	NA
9	2016-01-09T00:00:00	135	0	47	56	33	33	220	58	1	NA
10	2016-01-10T00:00:00	46	25	-11	33	-88	80	320	139	1	1
11	2016-01-11T00:00:00	20	43	-101	-88	-121	55	270	107	1	1
12	2016-01-12T00:00:00	20	41	-83	-49	-105	81	270	125	1	1
13	2016-01-13T00:00:00	8	13	-101	-77	-110	59	230	98	1	NA

*Fig 4: Shifting the Average Wind (**AWND**) column.*

6.2 - Formatting the data

There are still many issues with the data that we will deal with. The first is that most of the values in the table are in tenths of a unit. This is done so that all the number can appear as integers in the data table, which is not what we want. So, a ***TMAX*** of **-88** really means **-8.8** degree Celsius and a ***PRCP*** of **135** means **13.5** millimeters.

The columns that are presented in tenths of a unit are: **PRCP, SNOW, TAVG, TMAX, TMIN, AWND**, and **WSF2** -- or column numbers 2,3,4,5,6,7, and 9

We want to divide every value in these column by 10 so that the values are in standard units.

The following code takes all the values in columns 2,3,4,5,6,7,9 in ***lansingWeatherRS***, divides the values by 10, and save the results back to the same columns in ***lansingWeatherRS***.

```
1  lansingWeatherRS[,c(2,3,4,5,6,7,9)] = lansingWeatherRS[,c(2,3,4,5,6,7,9)]/10;
```

as a shortcut, you can use the sequence 2:7 to replace 2,3,4,5,6,7

```
1  lansingWeatherRS[,c(2:7,9)] = lansingWeatherRS[,c(2:7,9)]/10;
```

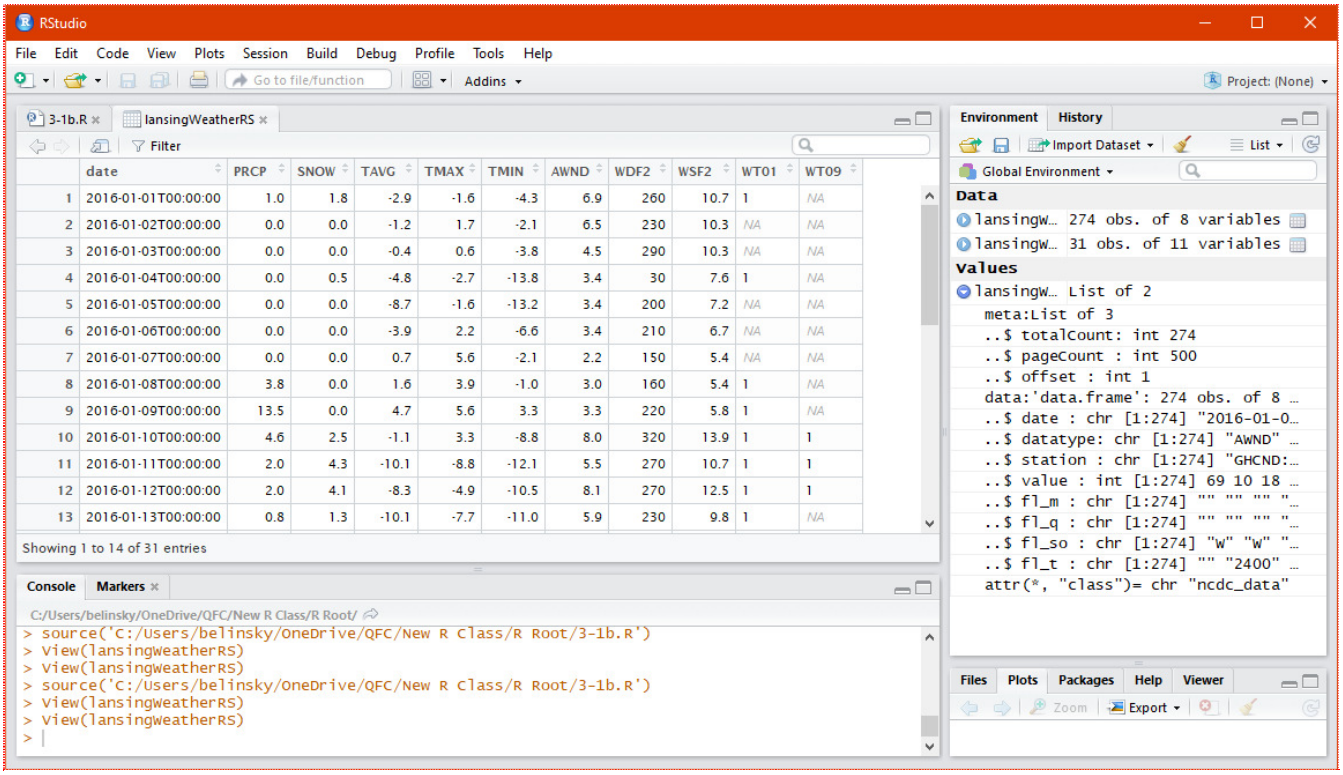


Fig 5: Changing from tenths of a unit to standard units.

6.3 - Conversions

The temperature values look much more sensible now but the temperatures are in Celsius and I am used to Fahrenheit (apologies to scientists and every country outside the United States -- this is just being used as an example). So, I would like to convert every value in **TAVG, TMIN**, and **TMAX** from Celsius to Fahrenheit.

The conversion is: **FahrenheitTemp = (9/5) * CelsiusTemp +32**
and we want to convert the values in columns 4, 5, and 6 *from all rows*.

```
1  lansingWeatherRS[,c(4:6)] = (9/5) * lansingWeatherRS[,c(4:6)] + 32;
```

Like before, we are performing a mathematical operation on all values in columns 4, 5, and 6 on ***lansingWeatherRS*** and saving the results back to columns 4, 5, and 6 of ***lansingWeatherRS***.

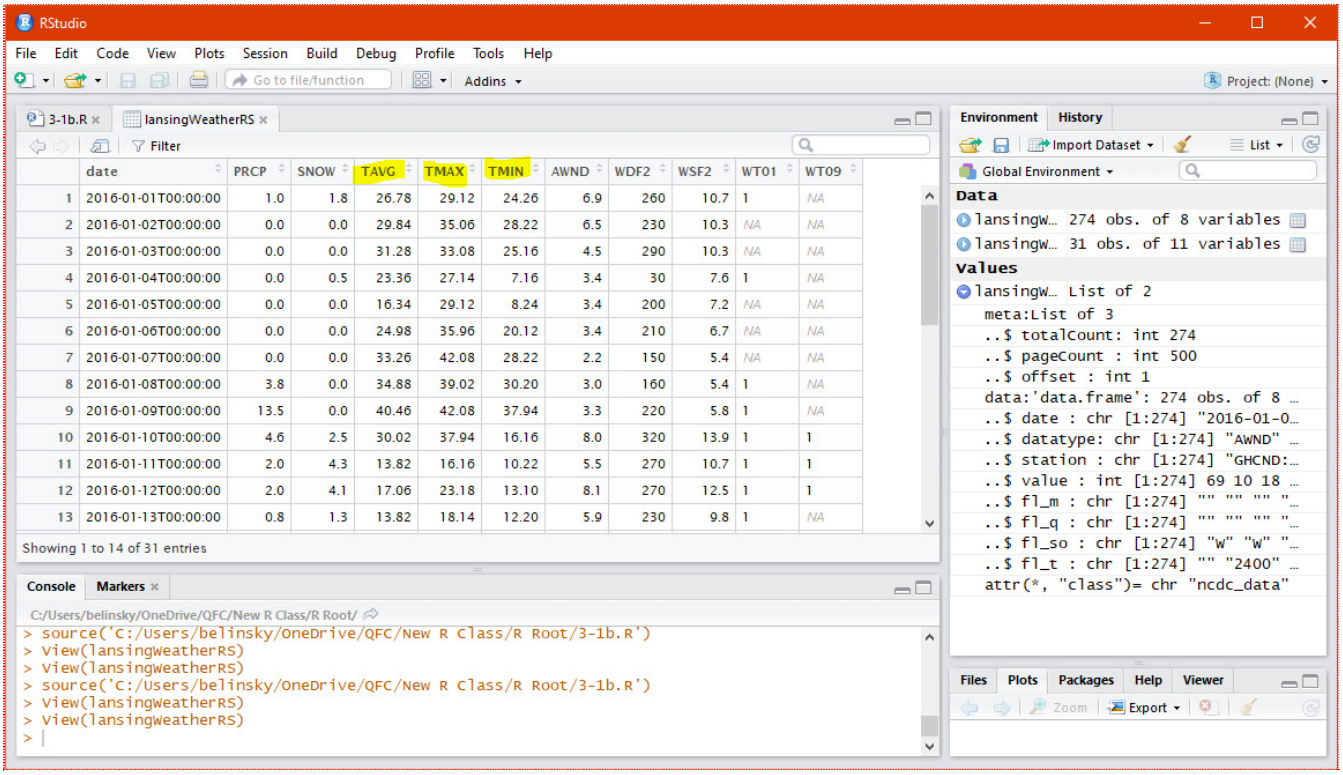


Fig 6: Converting temperature columns from Celsius to Fahrenheit (most scientists are cringing...).

6.4 - Rounding Values

I really have no need for 2 decimal places in the temperature value. I want just 1 decimal place. So, I will use the ***round()*** function to round all of the values in the temperature columns (4, 5, and 6) to one decimal place.

The parameters for ***round()*** are:

- ***x:*** the values to round
- ***digits***: where you want to round relative to the decimal point (negative numbers means go left of decimal)

```
1  lansingWeatherRS[,4:6] = round(x=lansingWeatherRS[,c(4,6)],digits=1);
```

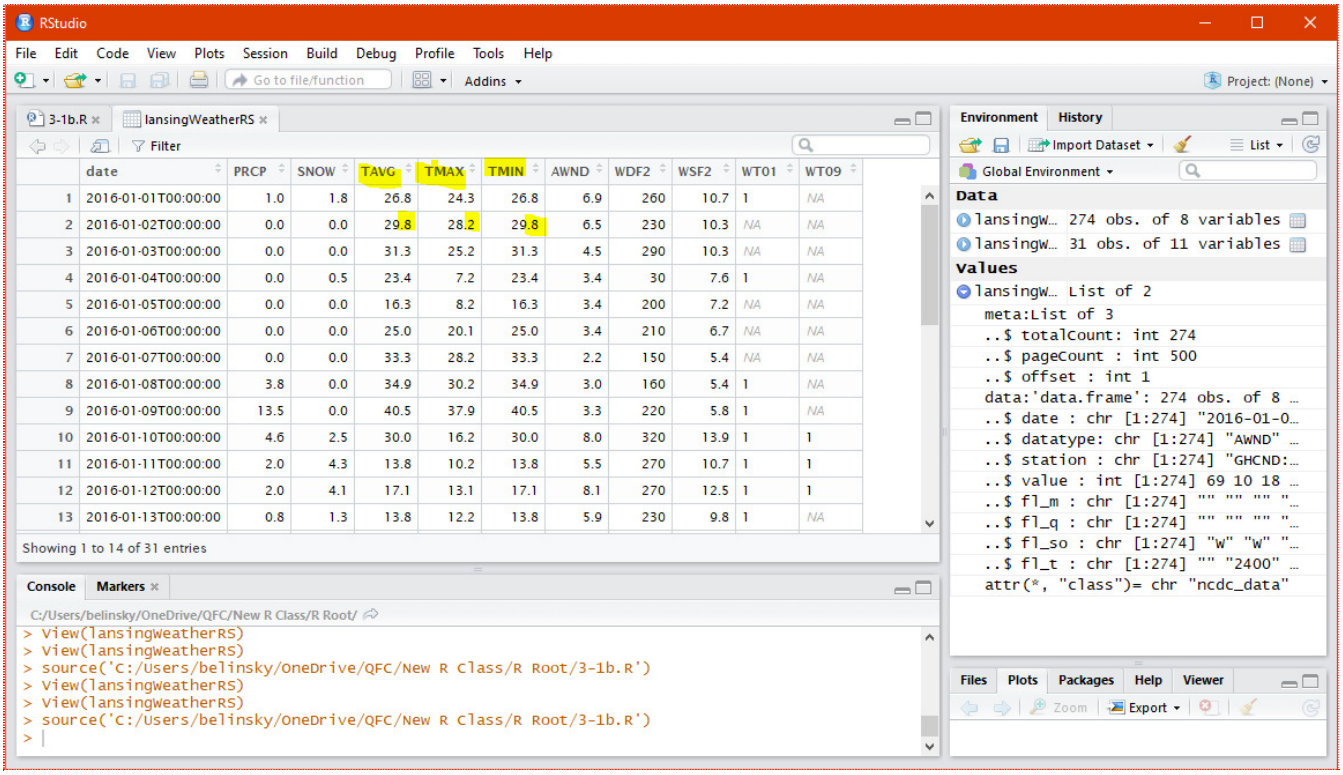


*Fig 7: Rounding temperature values using **round()**.*

6.5 - Getting a substring

I really don't want all the time information given in the ***date*** column (column 1). I really only need the two digit month and day, which is the *6th through the 10th characters* of each value. I can use the function ***substr()*** to pull out a portion of the string value and save that portion back to the column.

2016-***01-02***T00:00:00
2016-***01-14***T00:00:00
2016-***01-26***T00:00:00

substr() has three parameters that we are going to assign value to:

- ***x***: the values that we want to subset
- ***start***: the position we want the substring to start at
- ***end***: the position that we want the substring to end at

```
1  lansingWeatherRS[,1] = substr(x=lansingWeatherRS[,1], start=6, stop=10);
```

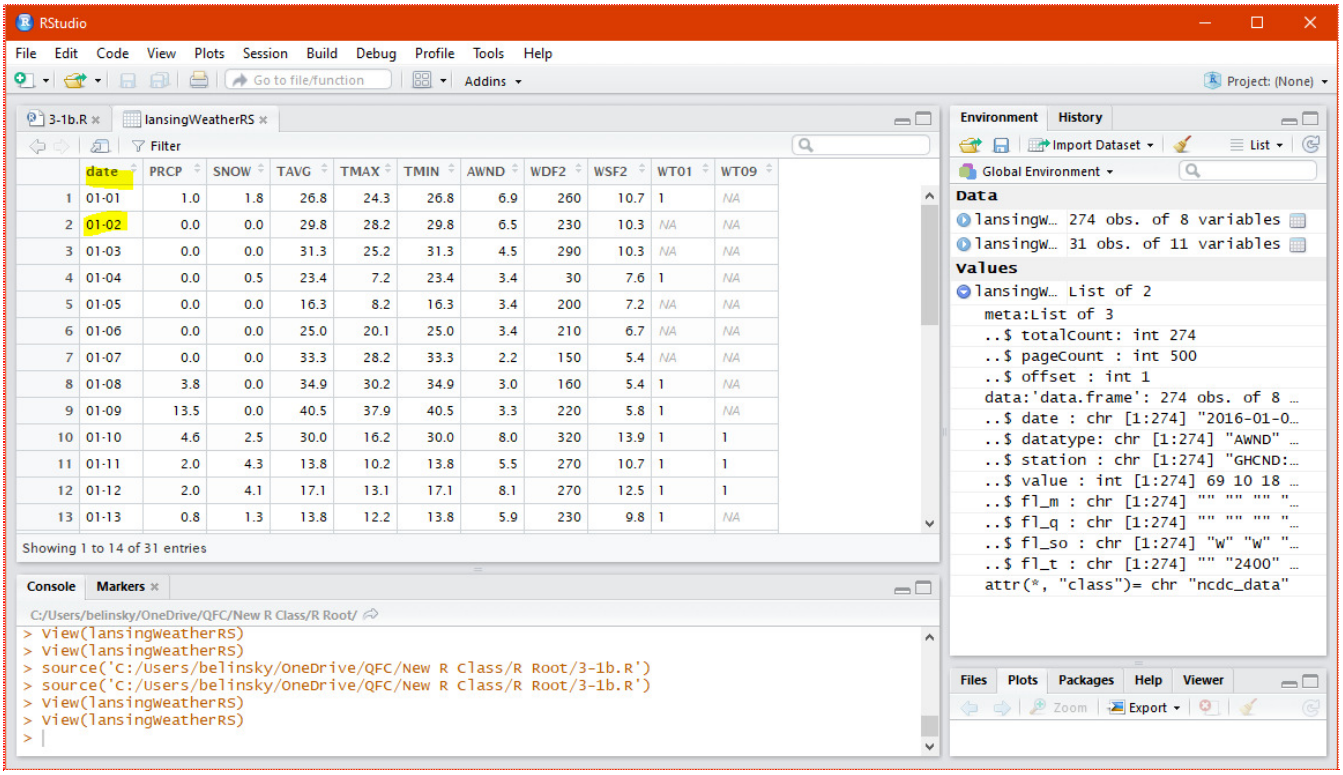


*Fig 8: Subset strings in the **date** column.*

6.6 - Changing column names

Lastly, some of the column have names that mean absolutely nothing to me, specifically column 10 (**WT01**) and column 11 (**WT09**). I would like to change the names to something more informative.

WT01 indicates if there was fog on that day, **WT09** indicates if there was blowing snow on that day.

We can use the function ***colnames()*** to change the column names. The only parameter we need to use is ***x***, which is the data frame. However, we don't want to change every column name so we need to index the columns 10 and 11.

```
1  colnames(x=lansingWeatherRS)[10] = "Fog";
2  colnames(x=lansingWeatherRS)[11] = "BSnow";
```

The above code changes column 10's name to "Fog" and column 11's name to "BSnow"

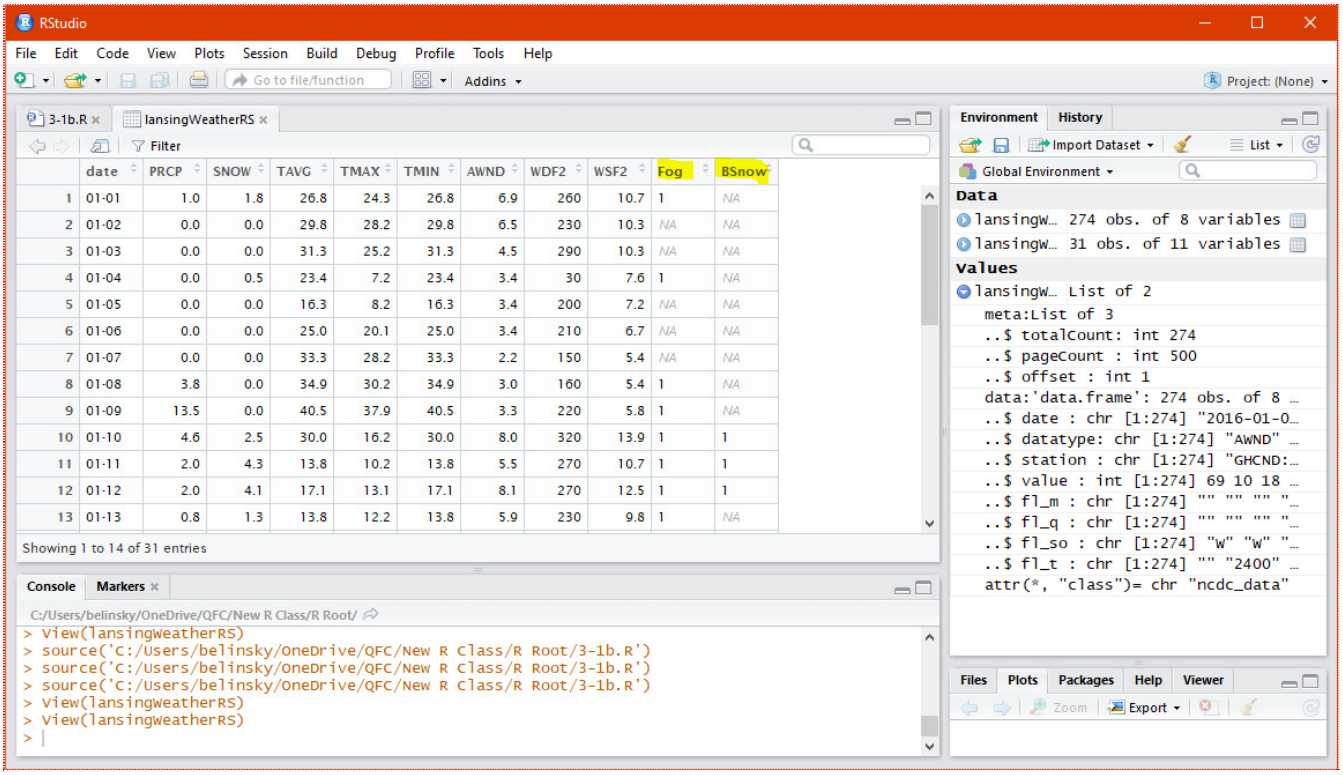


Fig 9: Changing the column names in the data frame.

7 - Saving the reformatted data frame

Our data frame is now in a clean and usable form. We don't want to have to go through this every time we use this data so let's save the reformatted data frame to a CSV file. That way, we can just open up the CSV file and go right to the formatted data.

We will use ***write.csv()*** to write the data frame to a CSV file. The parameters are:

- ***x***: the data frame to write
- ***file***: the file to write the data frame to

```
1  write.csv(x=lansingWeatherRS, file="data/formattedLansingWeather.csv");
```

The above code will create a file named ***formattedLansingWeather.csv*** in the ***data*** folder inside your ***R Root*** directory. ***formattedLansingWeather.csv*** can be opened by any R script and saved as a data frame (which we will do next lesson).

8 - Application

From last lesson:

1) Get the following data for December2016 in Lansing, MI: **PRCP, SNOW, TAVG, TMAX, TMIN**
2) Save the data frame (***data***) from the list to a variable named ***lansingDecWeather***

Now manipulate the data frame:

1) Reshape the data frame so ***date*** is on the x-axis (rows) and ***datatype*** is on the y-axis (columns)
2) Rearrange the columns to this order: **date, TMIN, TMAX, TAVG, PRCP, SNOW**
3) Rename the temperature columns to: **minTemp, maxTemp, avgTemp**
4) Divide all 5 data columns by 10 to get the values out of tenths of a unit.

5) Convert the **PRCP** and **SNOW** columns from mm to inches (25 millimeters equals 1 inch)
6) Substring the date column so that the dates look like this: **16-12-02**
7) Find the mean, standard deviation, and variance for the ***maxTemp*** and ***minTemp*** columns (use the ***mean()***, ***sd()***, and ***var()*** functions)
8) Challenge: Add a column to ***lansingDecWeather*** called ***changeTemp***. This column will hold the high temperature minus the low temperature for each day. Look at Unit 2, Lesson 5 for help.

9 - Extension: The four digit data type ids

The weather information we are getting in this lesson comes from the Global Historical Climate Network Database (GHCND), which is operated by NOAA/NCDC. The documentation for the GHCND is here. All the possible weather information collected in the GHCND is given on page 5, including the four-digit *data type id* for each (e.g., TMIN is minimum temperature, AWND is average wind). Note: each weather station collects a different set of information, so not every *data type id* is valid for a specific weather station.

10 - Extension: Functions with the same name

According to the Comprehensive R Archive Network (CRAN), there are over 12000 packages. This means that, inevitably, packages will reuse a name for a function. Or, perhaps you have created a function that has the same name as a function in a package. There are rules for how R determines which function gets executed if there are multiple functions with the same name, but it is easier to avoid the issue. You can do that by putting the package name in front of the function.

So, instead of making a call to ***dcast()***, which is in the ***reshape2*** package (and, hopefully is unique):

```
1  lansingWeatherRS = dcast(data = lansingWeatherDF,
2           formula = date ~ datatype,
3           value.var = "value");
```

We could be more explicit and specifically call ***dcast()*** from ***reshape2***. This is done by giving the package name and two semicolons:

```
1  lansingWeatherRS = reshape2::dcast(data = lansingWeatherDF,
2           formula = date ~ datatype,
3           value.var = "value");
```