03-03 Matrices

1 - Purpose

- Shape weather data into a two-dimensional structure (i.e., has rows and columns)
- Perform mathematics and simple statistics on the rows, columns, and whole structure
- Write the matrix to a file

2 - Class Project or other questions...

If you have any questions about your Class Project or the material in this lesson *feel free to email them to the instructor here*.

3 - Script for the lesson

The script for this lesson is located here. We will be going through the script in order. You might want to first comment out all the lines and uncomment the lines as you go through the lesson.

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.

4 - Two-dimensional data

A data frame is a two dimensional data structure that is a collection of related vectors; for instance different types of weather data collected at the same times, as shown in the last lesson.

Data frames, by design, have *columns that hold different types of data*. This means that mathematical or statistical operations can only be performed on (numeric) columns as it would not make sense to perform these operation across row (e.g., taking the mean of mixed precipitation and temperature values or columns that have string values like precipitation type).

However, we can make a two-dimensional structure where all of the values are the same type, called a **matrix**. In a matrix, you can perform mathematical and statistical operations across rows, down columns, or on the entire structure. The matrix we will use in this lesson contains temperatures values for all days in January over six years (2011 to 2016) (*Fig.1*).

	Jan 2011	**Jan 2012**	**Jan 2013**	**Jan 2014**	**Jan 2015**	**Jan 2016**
1	54	44	25	14	29	29
2	25	29	26	13	34	35
3	33	23	29	13	33	33
...	...	...	...			...
30	26	29	57	19	32	29
31	24	41	56	28	23	47

Fig 1: Partial matrix of January temperatures from 2011 through 2016 that we will create in this lesson

5 - The matrix

In the previous lesson, we got weather data from NOAA/NCDC and put it into a data frame. A data frame is a 2D data structure that contains multiple vectors of equal length. In this case the vectors represent different weather measurements like high temperature, wind speed, or precipitation. The rows of a data frame represent the instances of each vector -- in the case of our weather data, the rows represent days.

In this lesson, we will introduce the *matrix*, which is like a data frame except that all columns have the same type of value. A data frame can have factors or strings like weather conditions (e.g., snow, fog, or rain) in one column and numerical values like temperatures in another column whereas *all values in a matrix must be of one type*.

One advantage to putting data in a matrix is that R can perform statistical operations faster on a matrix than on a data frame -- this becomes noticeable as your data set gets larger or as you iterate procedures multiple times. Another advantage is that R can perform statistical and mathematical operation on rows of a matrix, columns of a matrix, or the whole matrix.

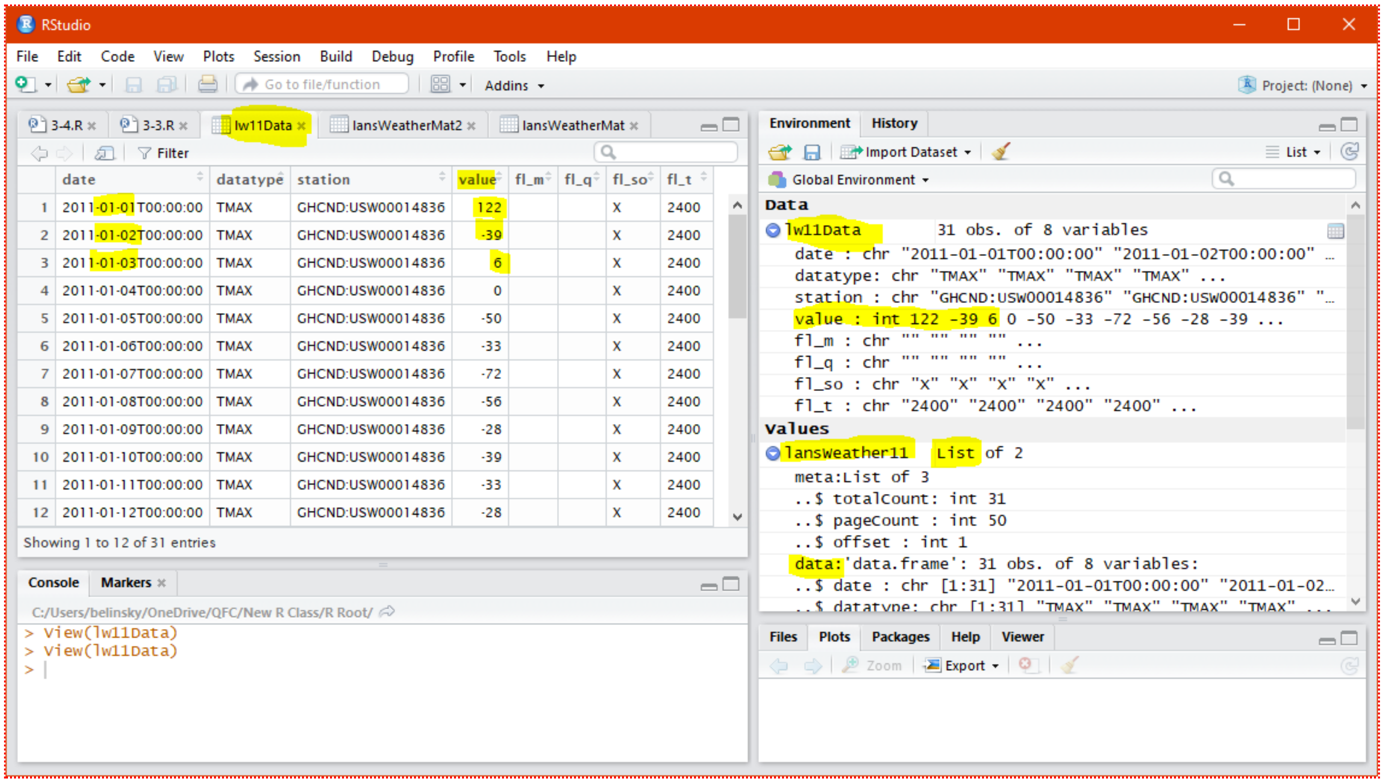
6 - Getting temperature data

We are going to get data from the NOAA/NCDC database. However, this time we are only getting one piece of data, the max temperature (***TMAX***) from all days in January, 2011.

```
lansWeather11 = ncdc(datasetid = "GHCND",
                     datatypeid = c("TMAX"),
                     stationid = "GHCND:USW00014836",
                     startdate = "2011-01-01", enddate ="2011-01-31",
                     token = myToken,
                     limit = 50 );
```

NOAA/NCDC returns a list object and we just want the data frame contained within the list. So, we need to subset the data frame from the list.

```
lansWeather11Data = lansWeather11[["data"]];
```



	date	datatype	station	value	fl_m	fl_q	fl_so	fl_t
1	2011-01-01T00:00:00	TMAX	GHCND:USW00014836	122			X	2400
2	2011-01-02T00:00:00	TMAX	GHCND:USW00014836	-39			X	2400
3	2011-01-03T00:00:00	TMAX	GHCND:USW00014836	6			X	2400
4	2011-01-04T00:00:00	TMAX	GHCND:USW00014836	0			X	2400
5	2011-01-05T00:00:00	TMAX	GHCND:USW00014836	-50			X	2400
6	2011-01-06T00:00:00	TMAX	GHCND:USW00014836	-33			X	2400
7	2011-01-07T00:00:00	TMAX	GHCND:USW00014836	-72			X	2400
8	2011-01-08T00:00:00	TMAX	GHCND:USW00014836	-56			X	2400
9	2011-01-09T00:00:00	TMAX	GHCND:USW00014836	-28			X	2400
10	2011-01-10T00:00:00	TMAX	GHCND:USW00014836	-39			X	2400
11	2011-01-11T00:00:00	TMAX	GHCND:USW00014836	-33			X	2400
12	2011-01-12T00:00:00	TMAX	GHCND:USW00014836	-28			X	2400

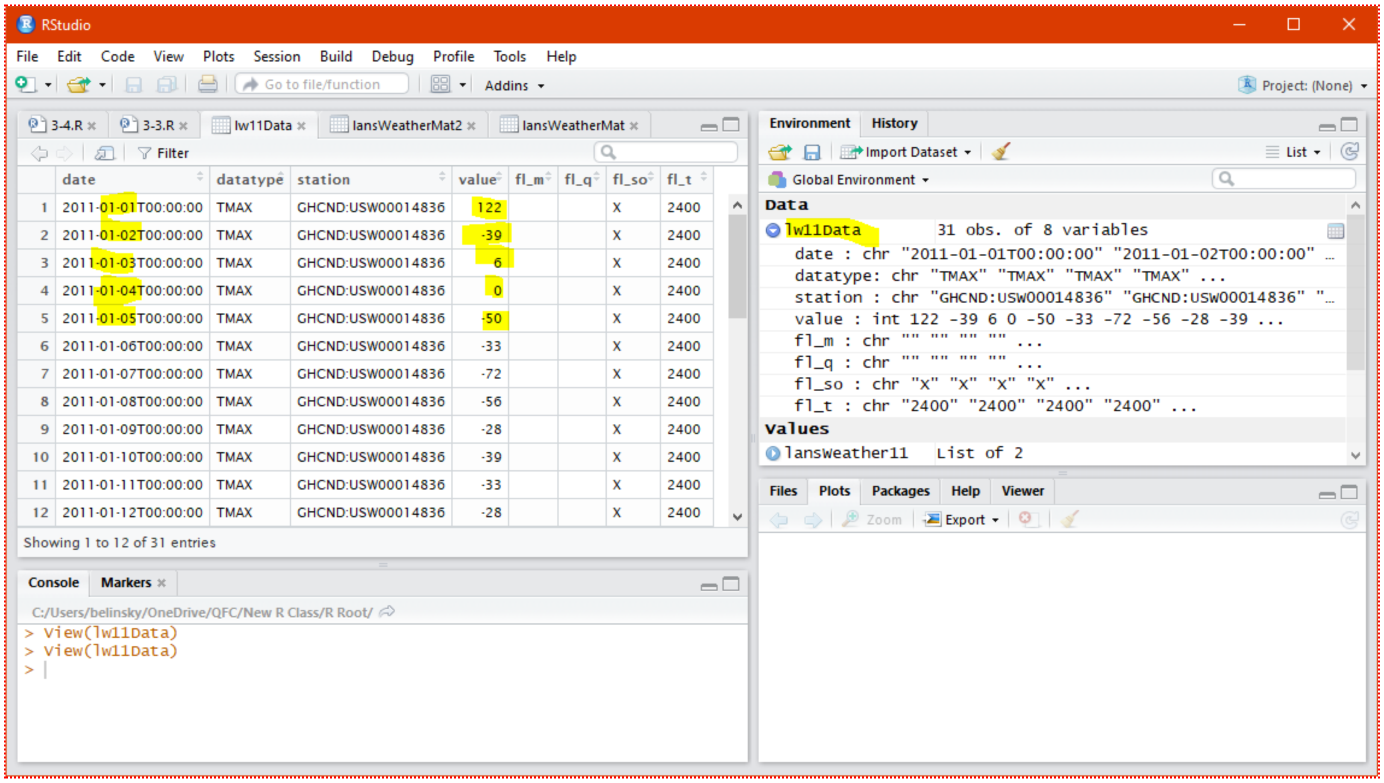
Showing 1 to 12 of 31 entries

```
Data
lw11Data          31 obs. of 8 variables
  date : chr "2011-01-01T00:00:00" "2011-01-02T00:00:00" ...
  datatype: chr "TMAX" "TMAX" "TMAX" "TMAX" ...
  station : chr "GHCND:USW00014836" "GHCND:USW00014836" "...
  value : int 122 -39 6 0 -50 -33 -72 -56 -28 -39 ...
  fl_m : chr "" "" "" "" ...
  fl_q : chr "" "" "" "" ...
  fl_so : chr "X" "X" "X" "X" ...
  fl_t : chr "2400" "2400" "2400" "2400" ...
Values
lansWeather11     List of 2
  meta:List of 3
  ..$ totalCount: int 31
  ..$ pageCount : int 50
  ..$ offset : int 1
  data:'data.frame': 31 obs. of 8 variables:
  ..$ date : chr [1:31] "2011-01-01T00:00:00" "2011-01-02...
  ..$ datatype: chr [1:31] "TMAX" "TMAX" "TMAX" "TMAX" ...
```

```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
> View(lw11Data)
> View(lw11Data)
> |
```

Fig 2: Getting the data from the list object returned by NOAA/NCDC's database

Because we only have one data type, the data frame is much simpler than last lesson. In this case, each row has a unique date and a maximum temperature (remember the temperature is in tenths of a Celsius degree -- we will take care of that in a bit).



	date	datatype	station	value	fl_m	fl_q	fl_so	fl_t
1	2011-01-01T00:00:00	TMAX	GHCND:USW00014836	122			X	2400
2	2011-01-02T00:00:00	TMAX	GHCND:USW00014836	-39			X	2400
3	2011-01-03T00:00:00	TMAX	GHCND:USW00014836	6			X	2400
4	2011-01-04T00:00:00	TMAX	GHCND:USW00014836	0			X	2400
5	2011-01-05T00:00:00	TMAX	GHCND:USW00014836	-50			X	2400
6	2011-01-06T00:00:00	TMAX	GHCND:USW00014836	-33			X	2400
7	2011-01-07T00:00:00	TMAX	GHCND:USW00014836	-72			X	2400
8	2011-01-08T00:00:00	TMAX	GHCND:USW00014836	-56			X	2400
9	2011-01-09T00:00:00	TMAX	GHCND:USW00014836	-28			X	2400
10	2011-01-10T00:00:00	TMAX	GHCND:USW00014836	-39			X	2400
11	2011-01-11T00:00:00	TMAX	GHCND:USW00014836	-33			X	2400
12	2011-01-12T00:00:00	TMAX	GHCND:USW00014836	-28			X	2400

Showing 1 to 12 of 31 entries

```
Data
lw11Data          31 obs. of 8 variables
  date : chr "2011-01-01T00:00:00" "2011-01-02T00:00:00" ...
  datatype: chr "TMAX" "TMAX" "TMAX" "TMAX" ...
  station : chr "GHCND:USW00014836" "GHCND:USW00014836" "...
  value : int 122 -39 6 0 -50 -33 -72 -56 -28 -39 ...
  fl_m : chr "" "" "" "" ...
  fl_q : chr "" "" "" "" ...
  fl_so : chr "X" "X" "X" "X" ...
  fl_t : chr "2400" "2400" "2400" "2400" ...
Values
lansWeather11     List of 2
```

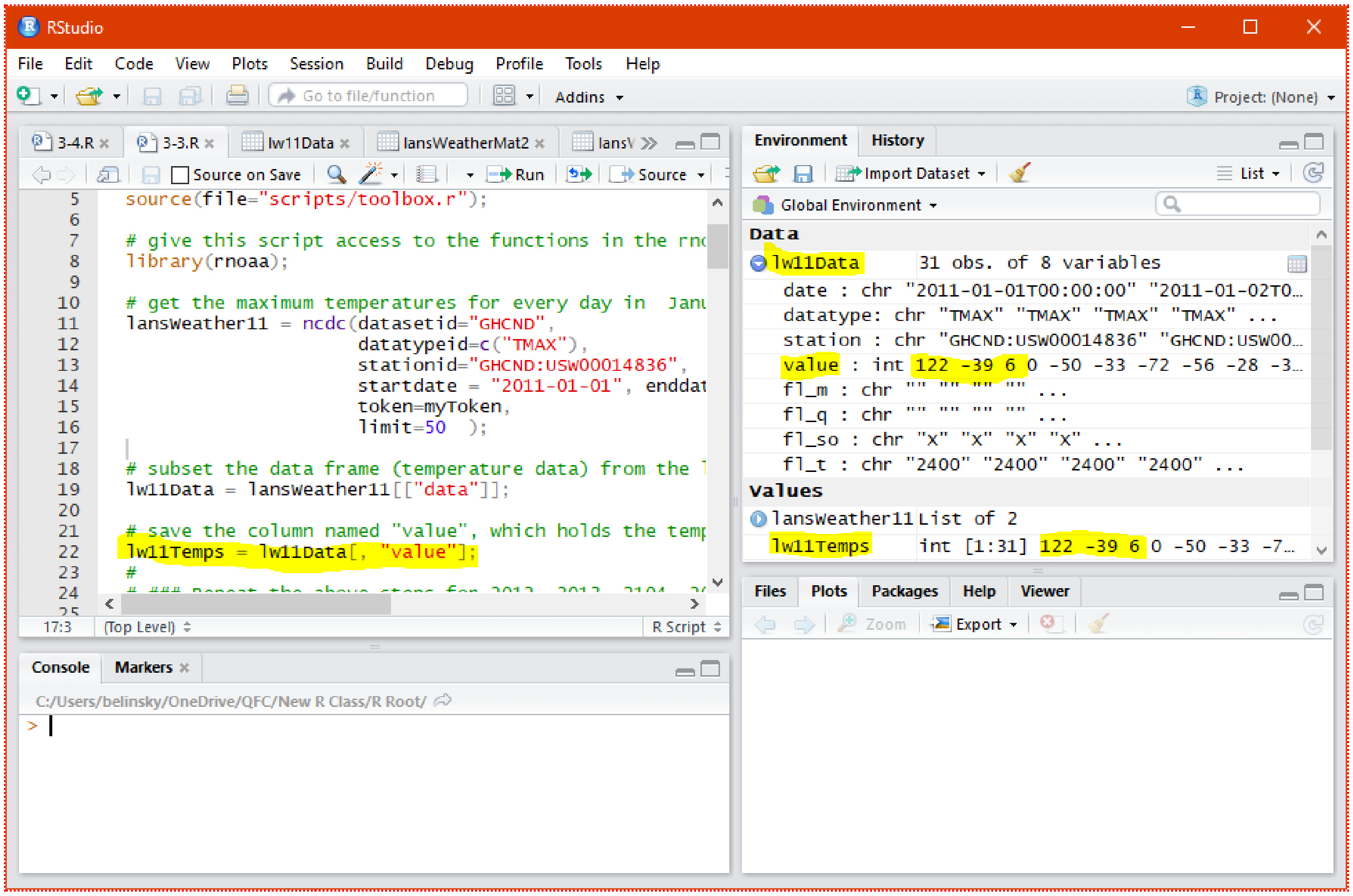
```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
> View(lw11Data)
> View(lw11Data)
> |
```

Fig 3: Data Frame showing unique date for each row and an associated temperature.

The ***value*** column holds the maximum temperatures, so we need to save the ***value*** column to a vector:

```
lansWeather11Val = lansWeather11Data$value;
```

Extension: dataframes and tibbles



```
source(file="scripts/toolbox.r");

# give this script access to the functions in the rno
library(rnoaa);

# get the maximum temperatures for every day in  Janu
lansWeather11 = ncdc(datasetid="GHCND",
                     datatypeid=c("TMAX"),
                     stationid="GHCND:USW00014836",
                     startdate = "2011-01-01", enddat
                     token=myToken,
                     limit=50  );

# subset the data frame (temperature data) from the l
lw11Data = lansWeather11[["data"]];

# save the column named "value", which holds the temp
lw11Temps = lw11Data[, "value"];
#
```

```
Data
lw11Data        31 obs. of 8 variables
  date : chr "2011-01-01T00:00:00" "2011-01-02T0…
  datatype: chr "TMAX" "TMAX" "TMAX" "TMAX" ...
  station : chr "GHCND:USW00014836" "GHCND:USW00…
  value : int 122 -39 6 0 -50 -33 -72 -56 -28 -3…
  fl_m : chr "" "" "" "" ...
  fl_q : chr "" "" "" "" ...
  fl_so : chr "x" "x" "x" "x" ...
  fl_t : chr "2400" "2400" "2400" "2400" ...
Values
lansWeather11   List of 2
lw11Temps       int [1:31] 122 -39 6 0 -50 -33 -7…
```

```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
>
```

Fig 4: Saving the "value" column from the data frame to a vector

6.1 - Repeat for all six years

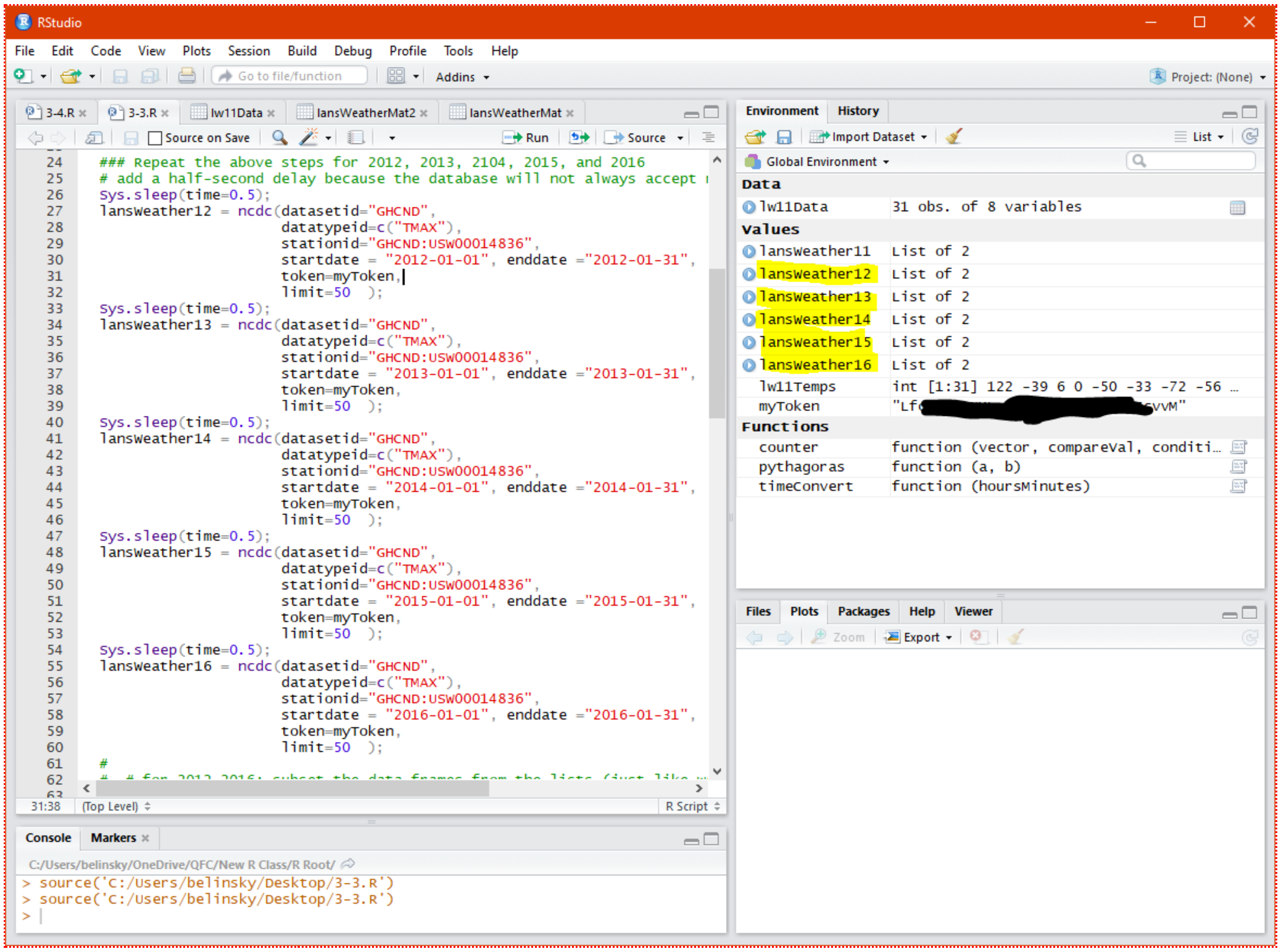
Now we are going to repeat the code above for the years 2012, 2013, 2014, 2015, and 2016. Note: *X* represents the last digit for the years 2012 to 2016.

```
lansWeather1X = ncdc(datasetid = "GHCND",
                     datatypeid  =c("TMAX"),
                     stationid  ="GHCND:USW00014836",
                     startdate = "201X-01-01", enddate ="201X-01-31",
                     token = myToken,
                     limit = 50 );
```

Sometimes the database does not accept multiple requests at once so we add a delay before each request.

We use ***Sys.sleep()*** to add a half second delay by using the parameter ***time*** = **0.5.**

```
Sys.sleep(time=0.5);
```



```
### Repeat the above steps for 2012, 2013, 2104, 2015, and 2016
# add a half-second delay because the database will not always accept
sys.sleep(time=0.5);
lansWeather12 = ncdc(datasetid="GHCND",
                     datatypeid=c("TMAX"),
                     stationid="GHCND:USW00014836",
                     startdate = "2012-01-01", enddate ="2012-01-31",
                     token=myToken,
                     limit=50  );
sys.sleep(time=0.5);
lansWeather13 = ncdc(datasetid="GHCND",
                     datatypeid=c("TMAX"),
                     stationid="GHCND:USW00014836",
                     startdate = "2013-01-01", enddate ="2013-01-31",
                     token=myToken,
                     limit=50  );
sys.sleep(time=0.5);
lansWeather14 = ncdc(datasetid="GHCND",
                     datatypeid=c("TMAX"),
                     stationid="GHCND:USW00014836",
                     startdate = "2014-01-01", enddate ="2014-01-31",
                     token=myToken,
                     limit=50  );
sys.sleep(time=0.5);
lansWeather15 = ncdc(datasetid="GHCND",
                     datatypeid=c("TMAX"),
                     stationid="GHCND:USW00014836",
                     startdate = "2015-01-01", enddate ="2015-01-31",
                     token=myToken,
                     limit=50  );
sys.sleep(time=0.5);
lansWeather16 = ncdc(datasetid="GHCND",
                     datatypeid=c("TMAX"),
                     stationid="GHCND:USW00014836",
                     startdate = "2016-01-01", enddate ="2016-01-31",
                     token=myToken,
                     limit=50  );
#
```

```
Data
lw11Data        31 obs. of 8 variables
Values
lansWeather11   List of 2
lansWeather12   List of 2
lansWeather13   List of 2
lansWeather14   List of 2
lansWeather15   List of 2
lansWeather16   List of 2
lw11Temps       int [1:31] 122 -39 6 0 -50 -33 -72 -56 ...
myToken         "Lf[redacted]SvvM"
Functions
counter         function (vector, compareVal, conditi…
pythagoras      function (a, b)
timeConvert     function (hoursMinutes)
```

```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
> source('C:/Users/belinsky/Desktop/3-3.R')
> source('C:/Users/belinsky/Desktop/3-3.R')
>
```

Fig 5: Getting temperature data from the other 5 years (2012-2016) with a half-second delay between requests

7 - Forming the matrix

A matrix is a two-dimensional structure (i.e., has rows and columns) where every value is of the same type. A matrix can also be thought of as a two-dimensional vector. In fact, one way to create a matrix is to break, or reshape, a vector up into rows and columns (i.e., make it two-dimensional).

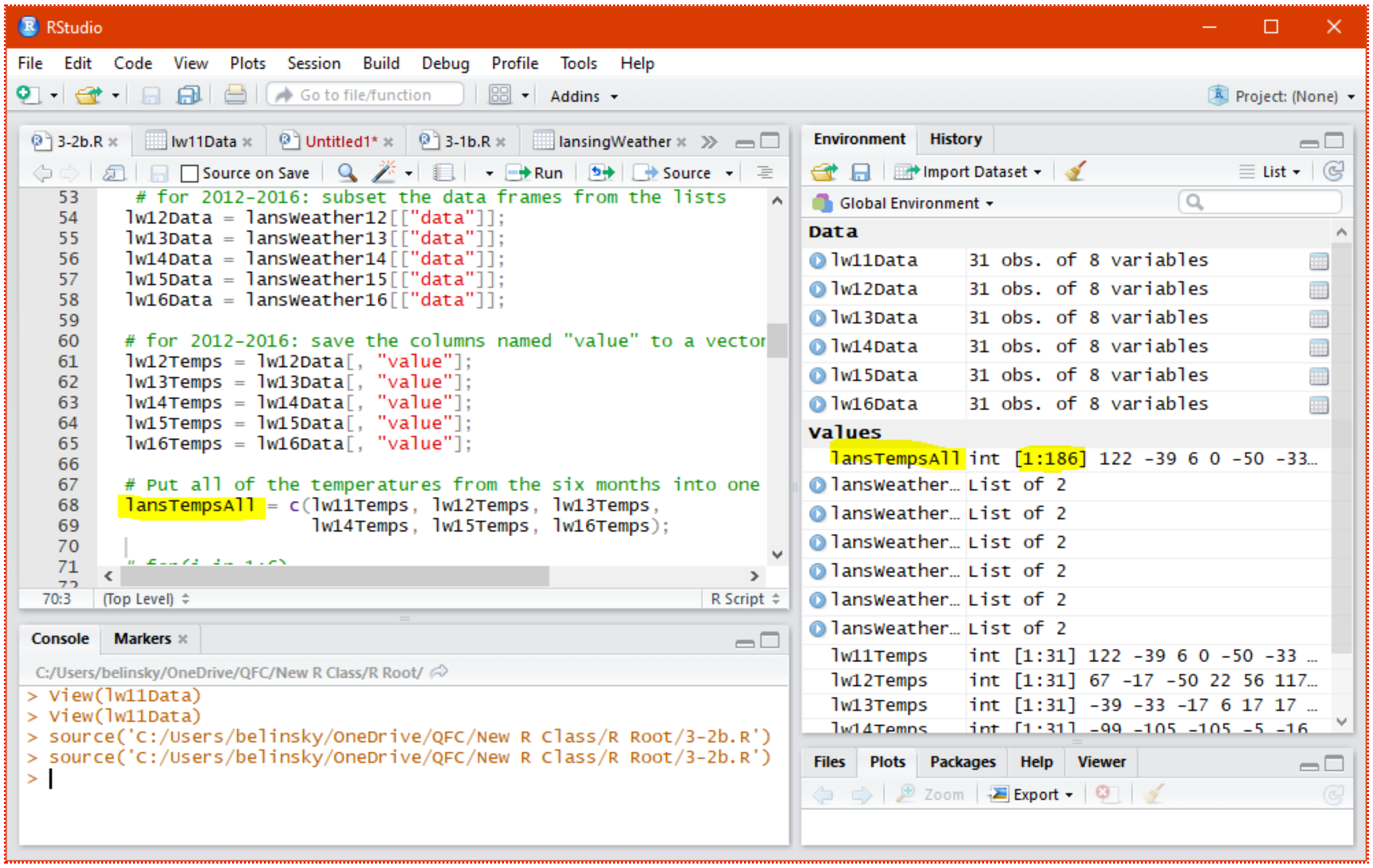
We are going to do this by:

- Putting all the January temperatures collected from NOAA/NCDC into one large vector
- Reshape the vector into rows and columns

7.1 - Create one large vector

Let's put the six months of temperatures vector into one large vector:

```
lansTempsAll = c(lw11Temps, lw12Temps, lw13Temps,
                 lw14Temps, lw15Temps, lw16Temps);
```



```
 # for 2012-2016: subset the data frames from the lists
lw12Data = lansWeather12[["data"]];
lw13Data = lansWeather13[["data"]];
lw14Data = lansWeather14[["data"]];
lw15Data = lansWeather15[["data"]];
lw16Data = lansWeather16[["data"]];

# for 2012-2016: save the columns named "value" to a vector
lw12Temps = lw12Data[, "value"];
lw13Temps = lw13Data[, "value"];
lw14Temps = lw14Data[, "value"];
lw15Temps = lw15Data[, "value"];
lw16Temps = lw16Data[, "value"];

# Put all of the temperatures from the six months into one
lansTempsAll = c(lw11Temps, lw12Temps, lw13Temps,
                 lw14Temps, lw15Temps, lw16Temps);
```

```
> View(lw11Data)
> View(lw11Data)
> source('C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/3-2b.R')
> source('C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/3-2b.R')
>
```

```
Data
lw11Data      31 obs. of 8 variables
lw12Data      31 obs. of 8 variables
lw13Data      31 obs. of 8 variables
lw14Data      31 obs. of 8 variables
lw15Data      31 obs. of 8 variables
lw16Data      31 obs. of 8 variables
Values
lansTempsAll  int [1:186] 122 -39 6 0 -50 -33…
lansweather…  List of 2
lansweather…  List of 2
lansweather…  List of 2
lansweather…  List of 2
lansweather…  List of 2
lansweather…  List of 2
lw11Temps     int [1:31] 122 -39 6 0 -50 -33 …
lw12Temps     int [1:31] 67 -17 -50 22 56 117…
lw13Temps     int [1:31] -39 -33 -17 6 17 17 …
lw14Temps     int [1:31] -99 -105 -105 -5 -16
```

Fig 6: Vector that hold all 6 months of temperatures.

7.2 - Reshape the vector into rows and columns

Now you have a vector with **31x6** = **186** values. Our next step is to create a matrix from these **186** values. To do this, we call the function ***matrix()***.

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

- ***data***: the vector that holds the values you want to put in the matrix
- ***nrow***: the number of rows
- ***ncol***: the number of columns
- ***byrow***: how the matrix is filled (***TRUE***: fill rows first, ***FALSE***: fill columns first)

In this case, the rows will be the days (***nrow=31***) and the columns will be the years (***ncol=6***). But how do we fill the cells? We can either go across rows first (***byrow* = TRUE**) or down columns first (***byrow* = FALSE**). The temperature vector, ***lansTempAll***, has all of January 2011 temperatures first, then all of January 2012 temperatures, etc. So, based on *Fig.7*, we want to go down columns first (***byrow*** = **FALSE**).

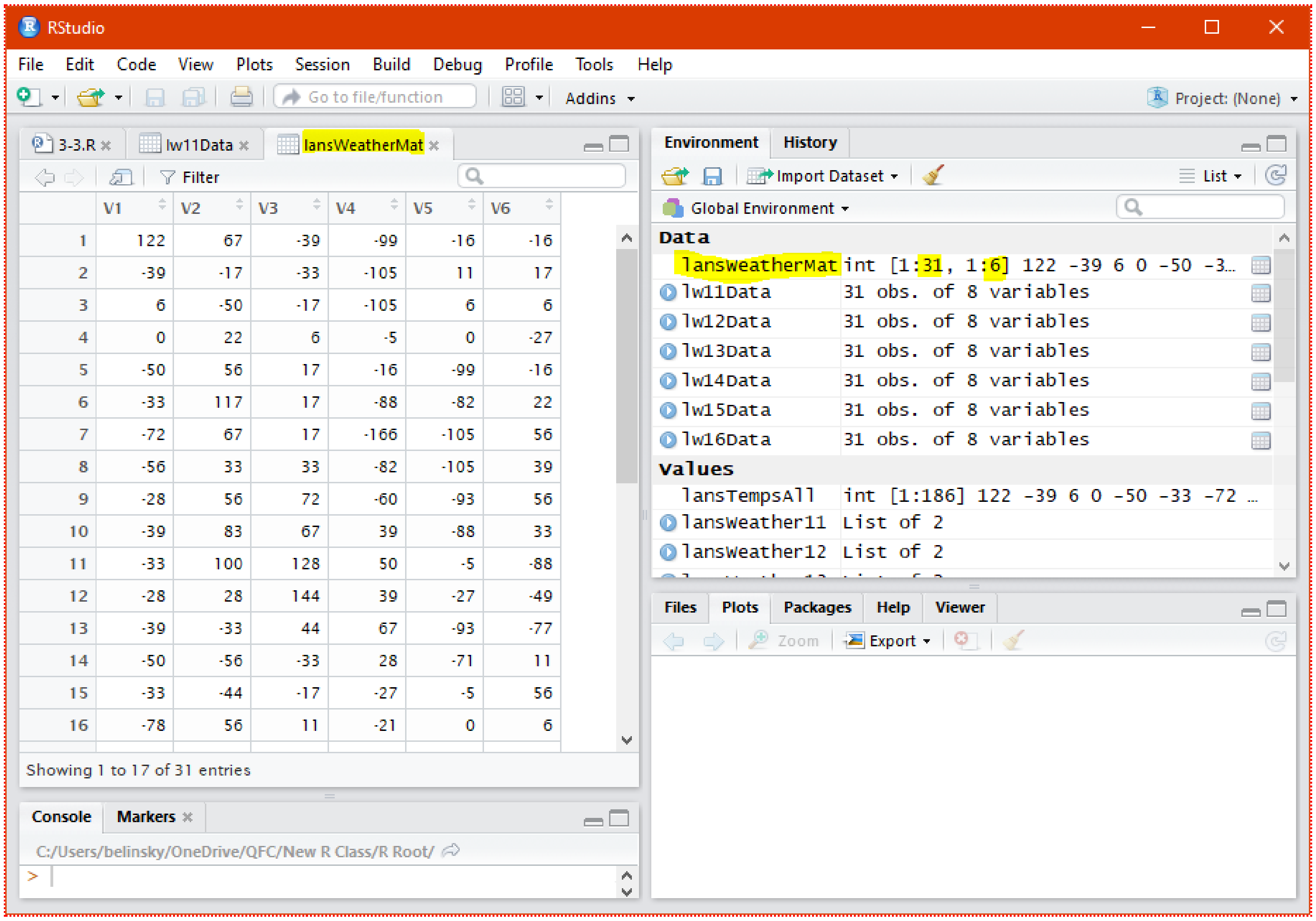
	Jan 2011	**Jan 2012**	**Jan2013**	**Jan2014**	**Jan 2015**	**Jan 2016**
1	*temp for Jan 1, 2011*	*temp for Jan 1, 2012*	*temp for Jan 1, 2013*			
2	*temp for Jan 2, 2011*					
3	*temp for Jan 3, 2011*					
…						
30						
31						

Fig 7: Mock-up of the two-dimensional table that will hold all the January temperatures.

So the call to ***matrix()*** looks like this:

```
lansWeatherMat = matrix(data=lansWeatherAll, nrow=31, ncol=6, byrow = FALSE);
```

And the matrix looks like this:



	V1	V2	V3	V4	V5	V6
1	122	67	-39	-99	-16	-16
2	-39	-17	-33	-105	11	17
3	6	-50	-17	-105	6	6
4	0	22	6	-5	0	-27
5	-50	56	17	-16	-99	-16
6	-33	117	17	-88	-82	22
7	-72	67	17	-166	-105	56
8	-56	33	33	-82	-105	39
9	-28	56	72	-60	-93	56
10	-39	83	67	39	-88	33
11	-33	100	128	50	-5	-88
12	-28	28	144	39	-27	-49
13	-39	-33	44	67	-93	-77
14	-50	-56	-33	28	-71	11
15	-33	-44	-17	-27	-5	56
16	-78	56	11	-21	0	6

Showing 1 to 17 of 31 entries

```
Data
 lansWeatherMat int [1:31, 1:6] 122 -39 6 0 -50 -3…
 lw11Data       31 obs. of 8 variables
 lw12Data       31 obs. of 8 variables
 lw13Data       31 obs. of 8 variables
 lw14Data       31 obs. of 8 variables
 lw15Data       31 obs. of 8 variables
 lw16Data       31 obs. of 8 variables
Values
 lansTempsAll   int [1:186] 122 -39 6 0 -50 -33 -72 …
 lansWeather11  List of 2
 lansWeather12  List of 2
```

```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
>
```

Fig 8: Matrix of 186 temperatures organized into 31 rows (days) and 6 columns (years).

*Extension: Using **cbind()** and **rbind()** to create matrices*

7.3 - More about the parameters of matrix()

The number of values in a matrix equals the number of rows times the number of columns or:
matrix.length = ncol * nrow

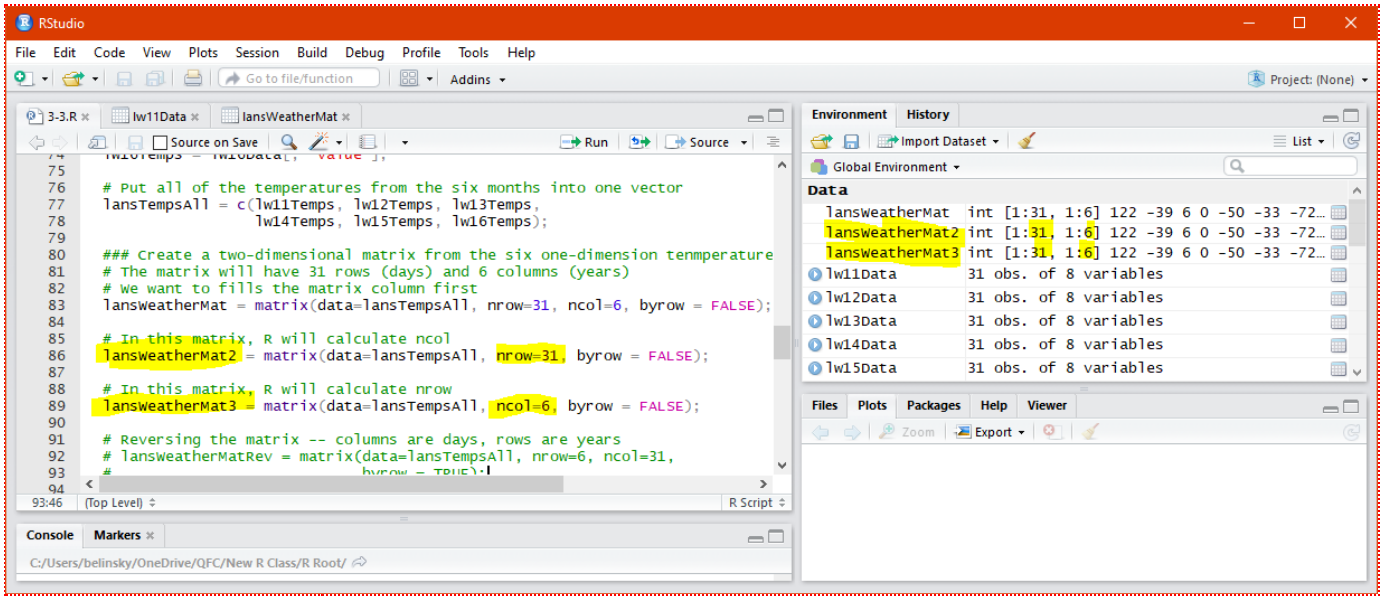
If R is only given two of the values, R will calculate the third.

In other words, you can skip ***ncol*** or ***nrow*** and R will calculate the missing value for you assuming the number of values is known. This is useful if you don't know how much data you have (e.g., you are getting an unknown number of years worth of temperatures). R will just keep adding new columns as more years of data are added.

```
lansWeatherMat2 = matrix(data=lansWeatherAll, nrow=31, byrow = FALSE);
```

And, while there is not an intuitive use for it in this example (you are not adding days to January!), you can skip ***nrow*** and R will calculate it for you.

```
lansWeatherMat3 = matrix(data=lansWeatherAll, ncol=6, byrow = FALSE);
```



```
lw16Temps = lw16Data[, "value"];

# Put all of the temperatures from the six months into one vector
lansTempsAll = c(lw11Temps, lw12Temps, lw13Temps,
                 lw14Temps, lw15Temps, lw16Temps);

### Create a two-dimensional matrix from the six one-dimension tenmperature
# The matrix will have 31 rows (days) and 6 columns (years)
# We want to fills the matrix column first
lansWeatherMat = matrix(data=lansTempsAll, nrow=31, ncol=6, byrow = FALSE);

# In this matrix, R will calculate ncol
lansWeatherMat2 = matrix(data=lansTempsAll, nrow=31, byrow = FALSE);

# In this matrix, R will calculate nrow
lansWeatherMat3 = matrix(data=lansTempsAll, ncol=6, byrow = FALSE);

# Reversing the matrix -- columns are days, rows are years
# lansWeatherMatRev = matrix(data=lansTempsAll, nrow=6, ncol=31,
#                            byrow = TRUE);
```

```
Data
  lansWeatherMat   int [1:31, 1:6] 122 -39 6 0 -50 -33 -72...
  lansWeatherMat2  int [1:31, 1:6] 122 -39 6 0 -50 -33 -72...
  lansWeatherMat3  int [1:31, 1:6] 122 -39 6 0 -50 -33 -72...
  lw11Data         31 obs. of 8 variables
  lw12Data         31 obs. of 8 variables
  lw13Data         31 obs. of 8 variables
  lw14Data         31 obs. of 8 variables
  lw15Data         31 obs. of 8 variables
```

*Fig 9: Having R calculate **nrow** or **ncol** -- the matrices are still the same.*

7.4 - Reversing the matrix

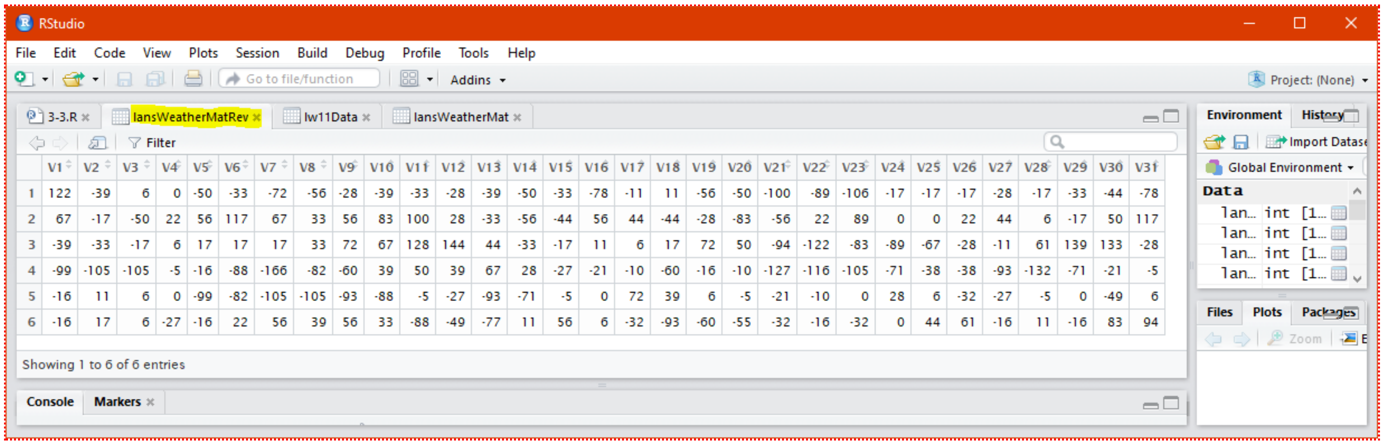
In the previous example (*Fig.9*), the rows represent days and the columns represent years. We could easily reverse the matrix so that the *rows represent the years* and the *columns represent the days*. Visualizing this:

	1	**2**	**3**	**...**	**30**	**31**
Jan 2011	Jan 1, 2011	Jan 2, 2011	Jan 3, 2011			
Jan 2012	Jan 1, 2012					
Jan 2013	Jan 1, 2013					
Jan 2014						
Jan 2015						
Jan 2016						

Fig 10: Mock-up of the REVERSED two-dimensional table that will hold all the January temperatures

The temperature vector, ***lansTempAll***, still has all of January 2011 temperatures first, then all of January 2012 temperatures, etc. So, based on *Fig.10*, we want to go *across rows first* (***byrow*** = **TRUE**).

```
lansWeatherMatRev = matrix(data=lansWeatherAll, nrow=6, ncol=31, byrow = TRUE);
```



	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
1	122	-39	6	0	-50	-33	-72	-56	-28	-39	-33	-28	-39	-50	-33	-78
2	67	-17	-50	22	56	117	67	33	56	83	100	28	-33	-56	-44	56
3	-39	-33	-17	6	17	17	17	33	72	67	128	144	44	-33	-17	11
4	-99	-105	-105	-5	-16	-88	-166	-82	-60	39	50	39	67	28	-27	-21
5	-16	11	6	0	-99	-82	-105	-105	-93	-88	-5	-27	-93	-71	-5	0
6	-16	17	6	-27	-16	22	56	39	56	33	-88	-49	-77	11	56	6

	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	V29	V30	V31
1	-11	11	-56	-50	-100	-89	-106	-17	-17	-17	-28	-17	-33	-44	-78
2	44	-44	-28	-83	-56	22	89	0	0	22	44	6	-17	50	117
3	6	17	72	50	-94	-122	-83	-89	-67	-28	-11	61	139	133	-28
4	-10	-60	-16	-10	-127	-116	-105	-71	-38	-38	-93	-132	-71	-21	-5
5	72	39	6	-5	-21	-10	0	28	6	-32	-27	-5	0	-49	6
6	-32	-93	-60	-55	-32	-16	-32	0	44	61	-16	11	-16	83	94

Showing 1 to 6 of 6 entries

```
Data
lan… int [1…
lan… int [1…
lan… int [1…
lan… int [1…
```

Fig 11: Transposing (i.e., reversing) the matrix: rows represent the 6 years and columns represent the 31 days

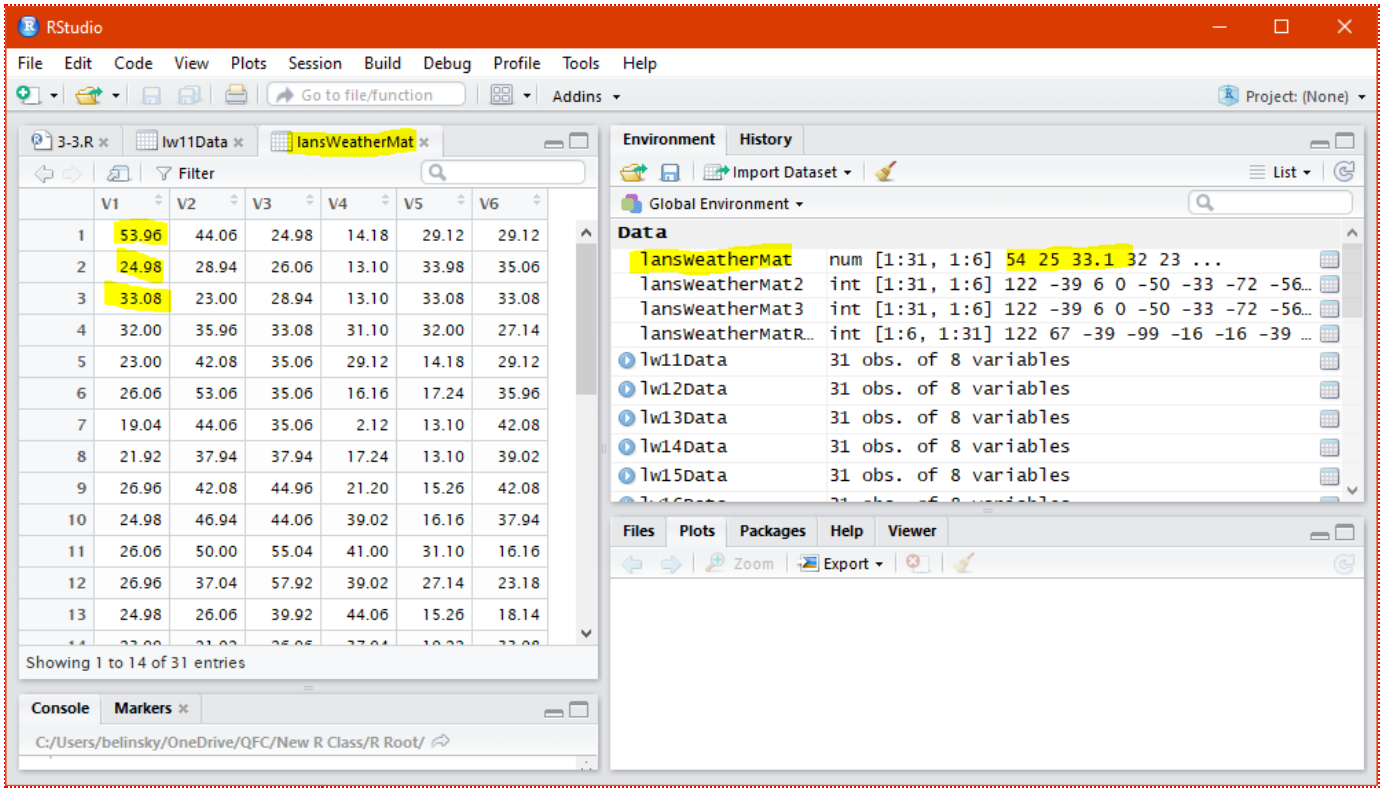
8 - Mathematics on the matrix

Let's go back to the original matrix with rows representing the days. Like last lesson, we have the issue that the temperatures are in tenths of a Celsius degree. To convert the temperatures to Fahrenheit we need to first divide the values by 10 and then convert from Celsius to Fahrenheit.

Since a matrix has all the same types of value, *we can apply these mathematical operations to the whole matrix* at once.

```
lansWeatherMat = lansWeatherMat * 0.1;
lansWeatherMat = (9/5) * lansWeatherMat + 32;
```

Note: This author is unsure why the values are rounded differently between the Environment Window and the matrix in the Main Window.



	V1	V2	V3	V4	V5	V6
1	53.96	44.06	24.98	14.18	29.12	29.12
2	24.98	28.94	26.06	13.10	33.98	35.06
3	33.08	23.00	28.94	13.10	33.08	33.08
4	32.00	35.96	33.08	31.10	32.00	27.14
5	23.00	42.08	35.06	29.12	14.18	29.12
6	26.06	53.06	35.06	16.16	17.24	35.96
7	19.04	44.06	35.06	2.12	13.10	42.08
8	21.92	37.94	37.94	17.24	13.10	39.02
9	26.96	42.08	44.96	21.20	15.26	42.08
10	24.98	46.94	44.06	39.02	16.16	37.94
11	26.06	50.00	55.04	41.00	31.10	16.16
12	26.96	37.04	57.92	39.02	27.14	23.18
13	24.98	26.06	39.92	44.06	15.26	18.14

Showing 1 to 14 of 31 entries

```
Data
lansWeatherMat      num [1:31, 1:6] 54 25 33.1 32 23 ...
lansWeatherMat2     int [1:31, 1:6] 122 -39 6 0 -50 -33 -72 -56…
lansWeatherMat3     int [1:31, 1:6] 122 -39 6 0 -50 -33 -72 -56…
lansWeatherMatR…    int [1:6, 1:31] 122 67 -39 -99 -16 -16 -39 …
lw11Data            31 obs. of 8 variables
lw12Data            31 obs. of 8 variables
lw13Data            31 obs. of 8 variables
lw14Data            31 obs. of 8 variables
lw15Data            31 obs. of 8 variables
```

C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/

Fig 12: Performing a mathematical operation on the whole matrix -- all temperatures are now in Fahrenheit.

8.1 - Significant Digits

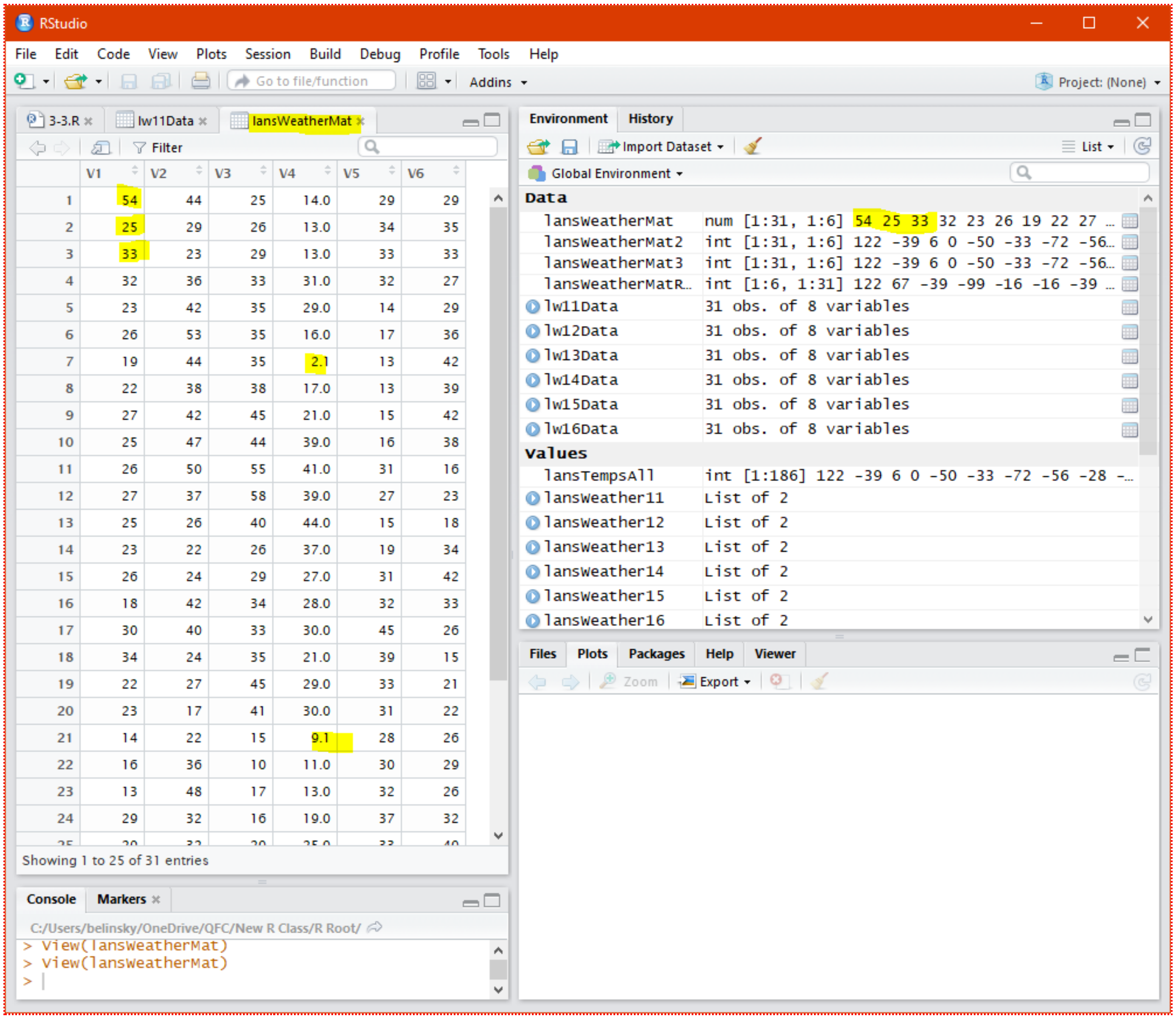
And, while we are at it, we only want two significant digits for each temperature (so, **53.96** becomes **54** and **9.14** become **9.1**). We can use the function ***signif()*** to set the significant digits.

signif() is similar to ***round()*** and has the parameters:

- ***x***: values to apply the significant digits to
- ***digits***: number of significant digits

```
lansWeatherMat = signif(x=lansWeatherMat, digits=2);
```

Again, R adds ***.0*** to all integer numbers in the 4th column. This is because the 4th column was the only column that had values less than 10, so the second significant digit was a decimal. If any column has any decimal number, all numbers are given a decimal. The ***.0*** is not significant, numerically speaking.



	V1	V2	V3	V4	V5	V6
1	54	44	25	14.0	29	29
2	25	29	26	13.0	34	35
3	33	23	29	13.0	33	33
4	32	36	33	31.0	32	27
5	23	42	35	29.0	14	29
6	26	53	35	16.0	17	36
7	19	44	35	2.1	13	42
8	22	38	38	17.0	13	39
9	27	42	45	21.0	15	42
10	25	47	44	39.0	16	38
11	26	50	55	41.0	31	16
12	27	37	58	39.0	27	23
13	25	26	40	44.0	15	18
14	23	22	26	37.0	19	34
15	26	24	29	27.0	31	42
16	18	42	34	28.0	32	33
17	30	40	33	30.0	45	26
18	34	24	35	21.0	39	15
19	22	27	45	29.0	33	21
20	23	17	41	30.0	31	22
21	14	22	15	9.1	28	26
22	16	36	10	11.0	30	29
23	13	48	17	13.0	32	26
24	29	32	16	19.0	37	32

Showing 1 to 25 of 31 entries

```
Data
   lansWeatherMat      num [1:31, 1:6] 54 25 33 32 23 26 19 22 27 ...
   lansWeatherMat2     int [1:31, 1:6] 122 -39 6 0 -50 -33 -72 -56...
   lansWeatherMat3     int [1:31, 1:6] 122 -39 6 0 -50 -33 -72 -56...
   lansWeatherMatR...  int [1:6, 1:31] 122 67 -39 -99 -16 -16 -39 ...
   lw11Data            31 obs. of 8 variables
   lw12Data            31 obs. of 8 variables
   lw13Data            31 obs. of 8 variables
   lw14Data            31 obs. of 8 variables
   lw15Data            31 obs. of 8 variables
   lw16Data            31 obs. of 8 variables
Values
   lansTempsAll        int [1:186] 122 -39 6 0 -50 -33 -72 -56 -28 -...
   lansWeather11       List of 2
   lansWeather12       List of 2
   lansWeather13       List of 2
   lansWeather14       List of 2
   lansWeather15       List of 2
   lansWeather16       List of 2
```

```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
> View(lansWeatherMat)
> View(lansWeatherMat)
> 
```

Fig 13: Setting the number of significant digits for the matrix.

9 - Statistics on the matrix

We can also apply statistical functions to the whole matrix. For instance, we can find the mean of all values in the matrix:

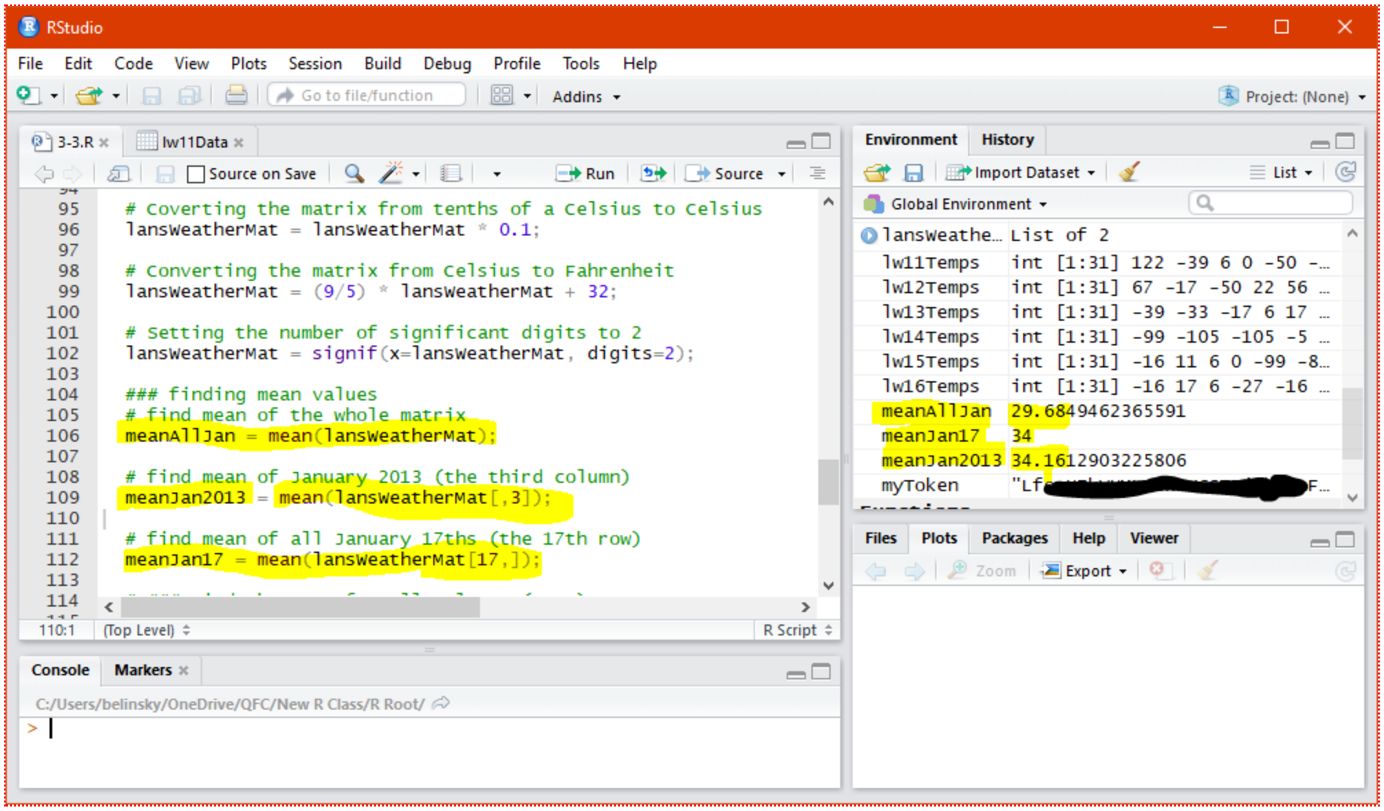
```r
meanAllTemps = mean(lansWeatherMat);
```

We can find the mean of a specific column of the matrix, representing the mean of a certain year:

```r
meanJan2013 = mean(lansWeatherMat[ ,3]);
```

Or find the mean of a specific row, representing all temperatures on a specific day in January:

```r
meanJan17 = mean(lansWeatherMat[17,]);
```



```r
# Coverting the matrix from tenths of a Celsius to Celsius
lansWeatherMat = lansWeatherMat * 0.1;

# Converting the matrix from Celsius to Fahrenheit
lansWeatherMat = (9/5) * lansWeatherMat + 32;

# Setting the number of significant digits to 2
lansWeatherMat = signif(x=lansWeatherMat, digits=2);

### finding mean values
# find mean of the whole matrix
meanAllJan = mean(lansWeatherMat);

# find mean of January 2013 (the third column)
meanJan2013 = mean(lansWeatherMat[,3]);

# find mean of all January 17ths (the 17th row)
meanJan17 = mean(lansWeatherMat[17,]);
```

```
lansWeathe…   List of 2
 lw11Temps    int [1:31] 122 -39 6 0 -50 -…
 lw12Temps    int [1:31] 67 -17 -50 22 56 …
 lw13Temps    int [1:31] -39 -33 -17 6 17 …
 lw14Temps    int [1:31] -99 -105 -105 -5 …
 lw15Temps    int [1:31] -16 11 6 0 -99 -8…
 lw16Temps    int [1:31] -16 17 6 -27 -16 …
meanAllJan    29.6849462365591
meanJan17     34
meanJan2013   34.1612903225806
myToken       "Lf[redacted]F…
```

```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
>
```

Fig 14: Finding means across a row, down a column, and for the whole matrix.

As a reminder, matrices, like a data frames, have two dimensions (rows and columns). So, we use two values to subset the matrix: a row and a column. If there is no subset value, then that means we are taking all values.

So: ***[,3]*** means take all rows in column 3
and: ***[17,]*** means take all columns in row 17

9.1 - Performing a mean down each column

We are going to find the mean for all six years of January temperatures. The columns in ***lansWeatherMatrix*** each hold one January's worth of maximum temperatures. So we want to find six mean values, one for each column in the matrix.

We need to save the mean values to a vector. To do this, we create a variable, ***yearlyMean,*** as a vector that initially has no values.

```
yearlyMean = c(); # a vector that holds the 6 yearly mean temps
```

Then we will use a ***for()*** to iterate through the 6 columns (years). Remember that ***i*** will take on each of the six values in the sequence **1:6** in turn as the loop iterates.

```
for(i in 1:6)
{
}
```

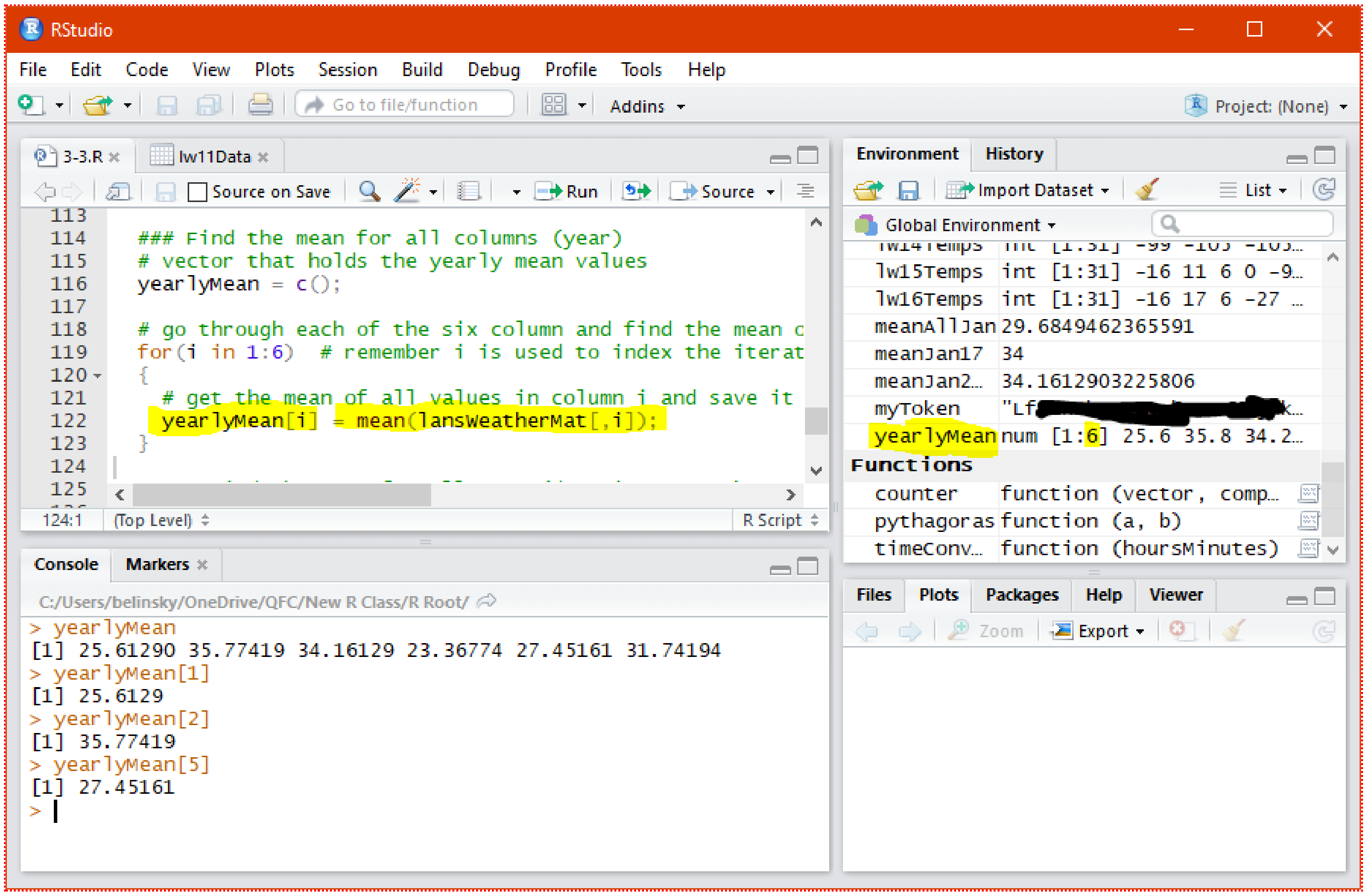
In the codeblock attached to the ***for(),*** we get the mean value of the temperatures in the indexed column: ***mean(JanTempMatrix[,i])***, where ***i*** will take on the values **1-6**.

Note: ***JanTempMatrix[,i]*** means get the value from every row in column ***i.***

We then assign the mean value to the ***yearlyMean*** vector: *yearlyMean[i]*

```
{
  yearlyMean = c();

  for(i in 1:6)
  {
    # get the mean of all values in column i and save it to monthlyMean[i]
    yearlyMean[i] = mean(lansWeatherMatrix[,i]);
  }
}
```

Once again, R displays numbers with different numbers of decimal places in different windows -- but the numbers are the same.



```
### Find the mean for all columns (year)
# vector that holds the yearly mean values
yearlyMean = c();

# go through each of the six column and find the mean c
for(i in 1:6)  # remember i is used to index the iterat
{
  # get the mean of all values in column i and save it
  yearlyMean[i] = mean(lansWeatherMat[,i]);
}
```

```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
> yearlyMean
[1] 25.61290 35.77419 34.16129 23.36774 27.45161 31.74194
> yearlyMean[1]
[1] 25.6129
> yearlyMean[2]
[1] 35.77419
> yearlyMean[5]
[1] 27.45161
>
```

```
lw14Temps   int [1:31] -99 -103 -103...
lw15Temps   int [1:31] -16 11 6 0 -9...
lw16Temps   int [1:31] -16 17 6 -27 ...
meanAllJan  29.6849462365591
meanJan17   34
meanJan2...  34.1612903225806
myToken     "Lf[redacted]k...
yearlyMean  num [1:6] 25.6 35.8 34.2...
Functions
counter     function (vector, comp...
pythagoras  function (a, b)
timeConv...  function (hoursMinutes)
```

*Fig 15: Getting the mean temperature for each month using a **for()***

9.2 - Performing a mean across all rows (days)

We can do the same thing across all rows to find the average temperature for each day in January across the six years.

In this case, we need to iterate the ***for()*** 31 times (**1:31**), and change the subset operation from

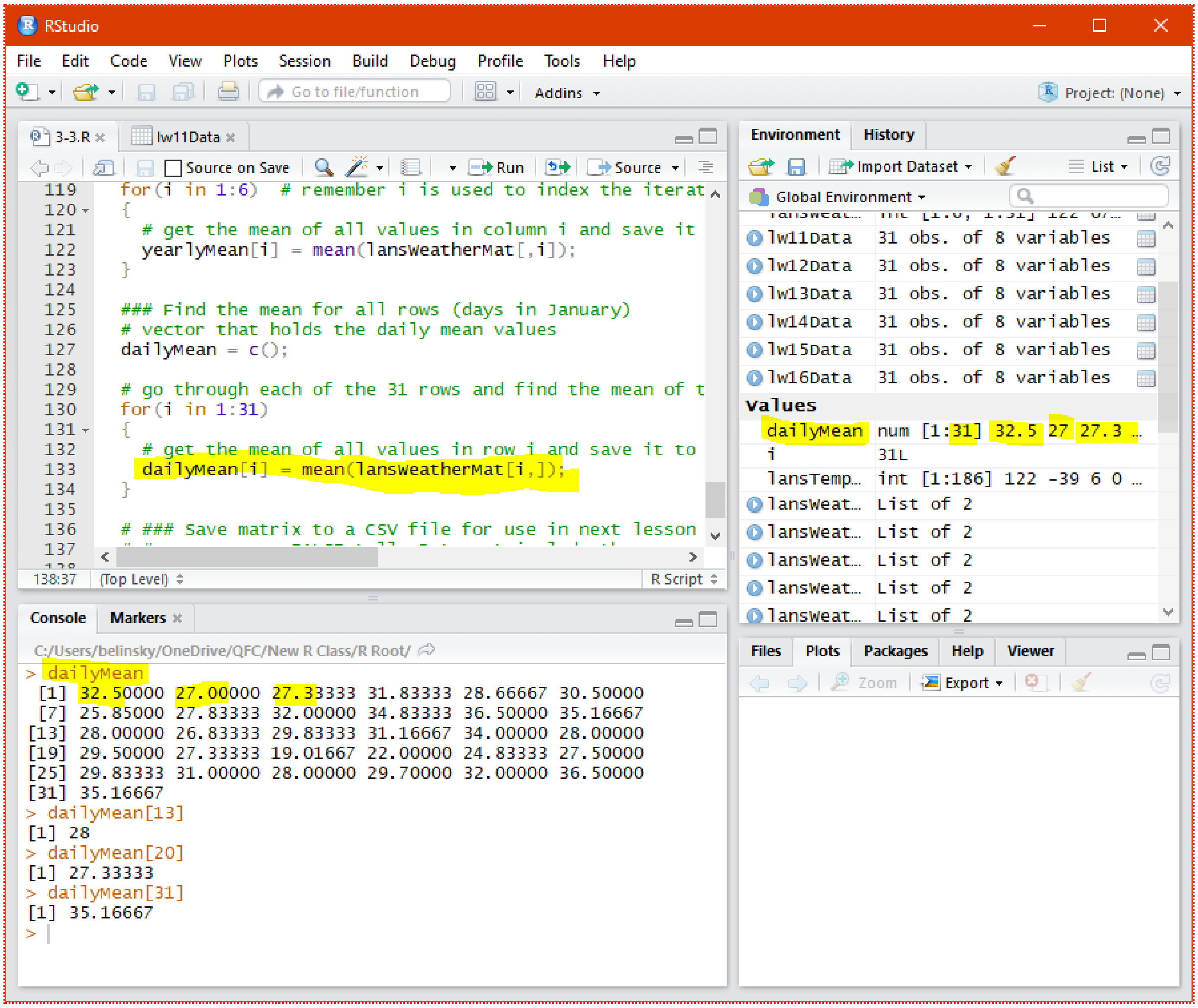
```
mean(lansWeatherMatrix[_,i]);   # get all row values from column i
```

to

```
mean(lansWeatherMatrix[i,_]);   # get all column values from row i
```

```
{
  dailyMean = c();

  for(i in 1:31)
  {
    # get the mean of all values in row i and save it to dailyMean[i]
    dailyMean[i] = mean(lansWeatherMatrix[i,]);
  }
}
```

We can use the Console Window to display any mean value. For example typing "dailyMean[13]" in the Console Window will output the **13th** value in ***dailyMean***.



```
for(i in 1:6)  # remember i is used to index the iterat
{
  # get the mean of all values in column i and save it
  yearlyMean[i] = mean(lansWeatherMat[,i]);
}

### Find the mean for all rows (days in January)
# vector that holds the daily mean values
dailyMean = c();

# go through each of the 31 rows and find the mean of t
for(i in 1:31)
{
  # get the mean of all values in row i and save it to
  dailyMean[i] = mean(lansWeatherMat[i,]);
}

# ### Save matrix to a csv file for use in next lesson
```

```
lw11Data     31 obs. of 8 variables
lw12Data     31 obs. of 8 variables
lw13Data     31 obs. of 8 variables
lw14Data     31 obs. of 8 variables
lw15Data     31 obs. of 8 variables
lw16Data     31 obs. of 8 variables
Values
dailyMean    num [1:31] 32.5 27 27.3 ...
i            31L
lansTemp...  int [1:186] 122 -39 6 0 ...
lansWeat...  List of 2
lansWeat...  List of 2
lansWeat...  List of 2
lansWeat...  List of 2
lansWeat...  List of 2
```

```
C:/Users/belinsky/OneDrive/QFC/New R Class/R Root/
> dailyMean
 [1] 32.50000 27.00000 27.33333 31.83333 28.66667 30.50000
 [7] 25.85000 27.83333 32.00000 34.83333 36.50000 35.16667
[13] 28.00000 26.83333 29.83333 31.16667 34.00000 28.00000
[19] 29.50000 27.33333 19.01667 22.00000 24.83333 27.50000
[25] 29.83333 31.00000 28.00000 29.70000 32.00000 36.50000
[31] 35.16667
> dailyMean[13]
[1] 28
> dailyMean[20]
[1] 27.33333
> dailyMean[31]
[1] 35.16667
> 
```

*Fig 16: Getting the mean temperature for each day in January using **for()***

10 - Saving the matrix

We are going to save ***lansWeatherMatrix*** to a CSV file so we can use it in the next lesson and avoid the hassle of calling NOAA/NCDC and reformatting the data.

```
write.csv(x=lansWeatherMat, file = "data/LansingJanTemps.csv", row.names = FALSE);
```

Now you should have a file called ***LansingJanTemps.csv*** in the ***data*** folder inside your ***R Root*** directory. Make sure you this file as you will be using it next lesson.

11 - Application

If you have any questions regarding your Class Project or this application, feel free to email them to the instructor here. You can attach the whole Root Folder as a zipped file.

1) Get the wind speed from the NOAA/NCDC database for January, February, March, and April 2016 in Lansing, MI.
- wind speed is ***datatypeid***: **AWND**
- query each month separately

2) Save the wind speeds into 4 vectors, each holding one months worth of wind values
3) Make all vectors equal in length (31) by added NA values to the end of February and April.
4) Combine the four vectors into one vector.
5) Create a matrix from the combined vector that has the months as columns and the days as rows:

	January	February	March	April
1				
2				
...				
30		NA		
31		NA		NA

6) Find the mean, standard deviation, and variance for the wind speed for February and March
- you will need to use the parameter ***na.rm*** to deal with the ***NA*** values.

7) Find the month that has the highest mean wind speed.
8) Find the day that has the highest mean wind speed.

*Save you script file as **app3-3.r** in the **scripts** folder of your RStudio Project for the class.*

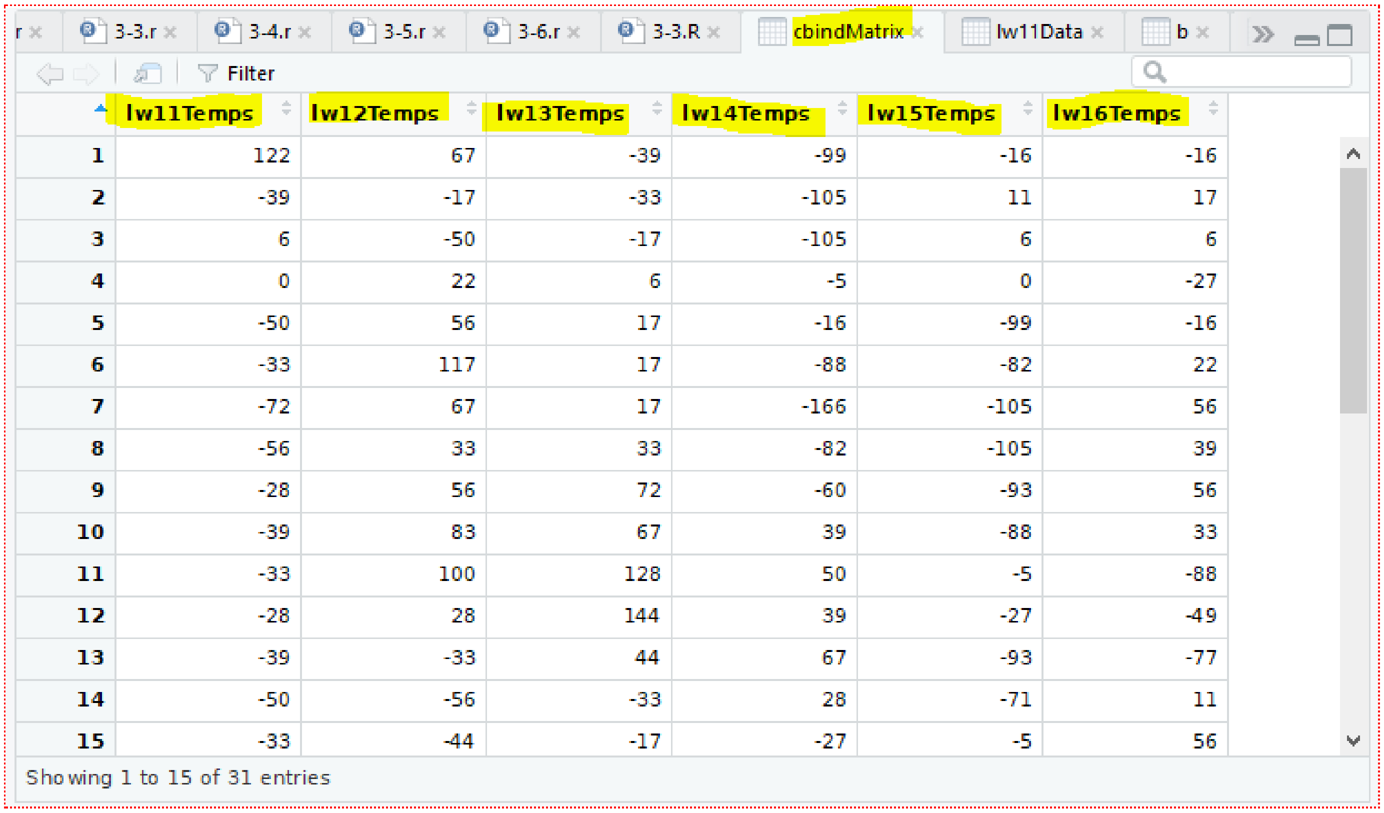
12 - Extension: Using cbind() and rbind() to create matrices

After the six calls to the NOAA/NCDC database, we extracted six vectors, each vector represented one January worth of maximum temperatures -- from 2011 to 2016.

In this lesson, we names those vector variables: ***lw11Temps, lw12Temps, lw13Temps, lw14Temps, lw15Temps***, and ***lw16Temps***

For the data in this lesson, there is a quick way to turn these six vector into a matrix and that is using ***cbind()*** or ***rbind().*** ***cbind()*** binds vectors together into a matrix placing each vector in a new column. ***rbind()*** binds vectors together into a matrix placing each vector in a new row. Effectively, ***rbind()*** created the transpose matrix of ***cbind().***

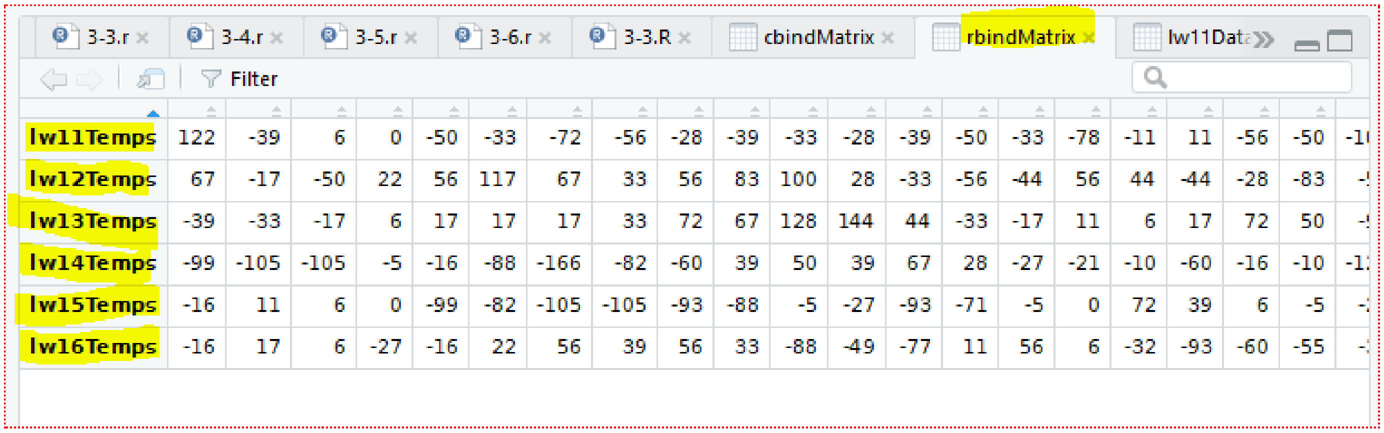
```
cbindMatrix = cbind(lw11Temps, lw12Temps, lw13Temps,
                    lw14Temps, lw15Temps, lw16Temps);

rbindMatrix = rbind(lw11Temps, lw12Temps, lw13Temps,
                    lw14Temps, lw15Temps, lw16Temps);
```



	lw11Temps	lw12Temps	lw13Temps	lw14Temps	lw15Temps	lw16Temps
1	122	67	-39	-99	-16	-16
2	-39	-17	-33	-105	11	17
3	6	-50	-17	-105	6	6
4	0	22	6	-5	0	-27
5	-50	56	17	-16	-99	-16
6	-33	117	17	-88	-82	22
7	-72	67	17	-166	-105	56
8	-56	33	33	-82	-105	39
9	-28	56	72	-60	-93	56
10	-39	83	67	39	-88	33
11	-33	100	128	50	-5	-88
12	-28	28	144	39	-27	-49
13	-39	-33	44	67	-93	-77
14	-50	-56	-33	28	-71	11
15	-33	-44	-17	-27	-5	56

Showing 1 to 15 of 31 entries

Fig 17: ***cbind()*** *makes each vector a column in the new matrix*



lw11Temps	122	-39	6	0	-50	-33	-72	-56	-28	-39	-33	-28	-39	-50	-33	-78	-11	11	-56	-50	-1
lw12Temps	67	-17	-50	22	56	117	67	33	56	83	100	28	-33	-56	-44	56	44	-44	-28	-83	-
lw13Temps	-39	-33	-17	6	17	17	17	33	72	67	128	144	44	-33	-17	11	6	17	72	50	-
lw14Temps	-99	-105	-105	-5	-16	-88	-166	-82	-60	39	50	39	67	28	-27	-21	-10	-60	-16	-10	-1
lw15Temps	-16	11	6	0	-99	-82	-105	-105	-93	-88	-5	-27	-93	-71	-5	0	72	39	6	-5	-
lw16Temps	-16	17	6	-27	-16	22	56	39	56	33	-88	-49	-77	11	56	6	-32	-93	-60	-55	-

Fig 18: ***rbind()*** *makes each vector a row in the new matrix*

13 - Extension: dataframes and tibbles

In March 2019, the line of code to get a column of temperature values from the ***lansWeather11Data*** dataframe (and the other 5 years) changed from

```
lansWeather11Val = lansWeather11Data [, "value"];
```

to

```
lansWeather11Val = lansWeather11Data$value;
```

The reason for this change is that the NOAA/NCDC database starting sending data in tibbles instead of dataframes. Tibbles are designed to be a more modern take on the dataframe, basically taking the lessons from decades on dataframe usage to create a better dataframe.

Tibbles are essentially the tidyverse version of a data frame and they supposedly improve on the data frame.

But, one difference is that in a data frame the three following lines all function exactly the same – they each take the column ***value*** from the data frame ***lw11Data*** and save it to a vector called ***temps***:

```
temps=lw11Data$value
temps=lw11Data[, "value"]
temps=lw11Data[["value"]]
```

However, in a tibble, the following line saves the column ***value*** into *a new 1 column tibble* called ***temps***.

```
temps=lw11Data[,"value"]
```

In other words, ***temps*** is not a vector, hence the error.

These two lines work the same in tibbles and data frames:

```
temps=lw11Data$value

temps=lw11Data[[“value”]]
```