

03-05 Sampling

1 - Purpose

- Creating a uniform random sample
- Creating a normal sample
- Creating repeatable pseudo-random values
- Saving and loading list objects

2 - Class Project issues 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 and data for this lesson

[You can get the script for this lesson here](#). The lesson goes in order on the script. You can comment and uncomment multiple lines in the script using Control-Shift-C in R (Command-Shift-C on a MAC).

You will also need the data file, ***lansingJanTemps.csv***, created in lesson 3-3. If you do not have this file, [download it here](#) and *save it to your data folder*.

4 - Uniform random sampling

There are times when you want to take random samples from a large database of values, like temperatures from multiple years and locations. One possible reason is to evaluate a suite of potential multiple potential weather scenarios. In this example, we will take random values without any weighting. In other words, all values have the same probability of being chosen.

Note: this lesson involved code that picks random values. This means that the script will produce different results each time it is executed -- until we introduce ***set.seed()***.

4.1 - sample of numbers

We can use the ***sample()*** function to extract a random sample from a set of observations.

sample() has three parameters:

- ***x***: the observations we are taking a random sample from (in this case, a sequence vector)
- ***size***: the number of samples we will take.
- ***replace***: whether we allow for an observation to be sampled multiple times (***replace=TRUE*** means we allow resampling of the same observation)

In the following code, the parameters for ***sample()*** are set so that:

- The data set we are using is the sequence 1-10: ***c(1:10)***
- We are taking eight random samples: ***size=8***

- We allow repeat observations (**replace**) in **randomSample1** but not in **randomSample2**.

Note: different observations can still have the same value

```
1 randomSample1 = sample( c(1:10), size=8, replace=TRUE);
2 randomSample2 = sample( c(1:10), size=8, replace=FALSE);
```

In Fig. 1, **randomSample1** has two values that were repeated (3 and 5), whereas **randomSample2** has no repeated values. *Running the script again will produce different values* but **randomSample2** will always have unique values because **replace=FALSE**.

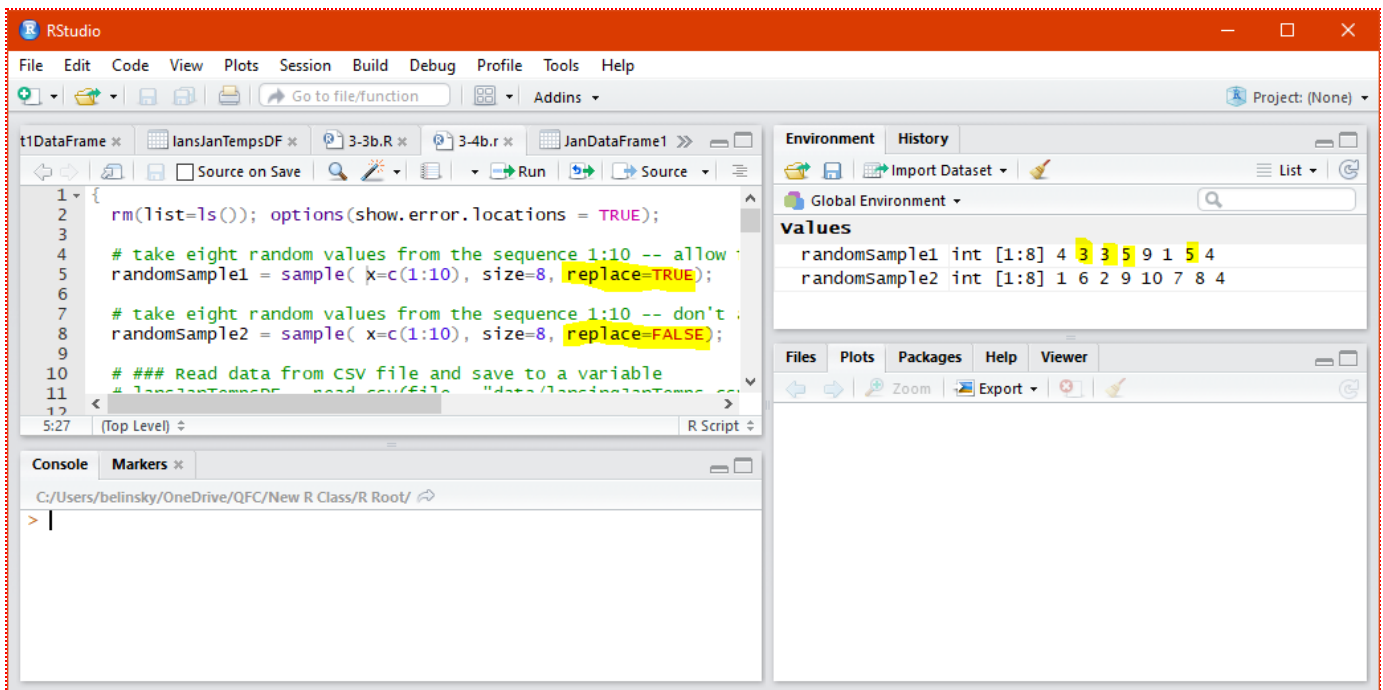


Fig 1: Random sampling of a vector with replacement and without replacement

Note: You will get an error if the parameter **size** is greater than the population size and **replace=FALSE**:

```
1 randomSample2 = sample( c(1:10), size=12, replace=FALSE); # will cause error
2 randomSample2 = sample( c(1:10), size=12, replace=TRUE); # will work
```

This is because you cannot take 12 **unique** samples from a population whose size is 10 (**replace=FALSE**). But you can take 12 samples from a population whose size is 10 (**replace=TRUE**) if you samples do not need to be unique.

4.2 - Sampling a data vector

We can also use **sample()** to choose random observations from a matrix. Remember that **a matrix is a reshaped vector with rows and columns**. In this case, we will get random observations from the January temperature matrix from the previous lessons.

First we will get the data from the CSV file we create in lesson 3-2 and then we need to convert the data frame into a matrix.

```
1 lansJanTempsDF = read.csv(file = "data/lansingJanTemps.csv");
2 lansJanTemps = as.matrix(x=lansJanTempsDF);
```

Remember that **lansJanTemps** contain 6 years worth of January temperature. So, $31 \times 6 = 186$ temperature observations in all.

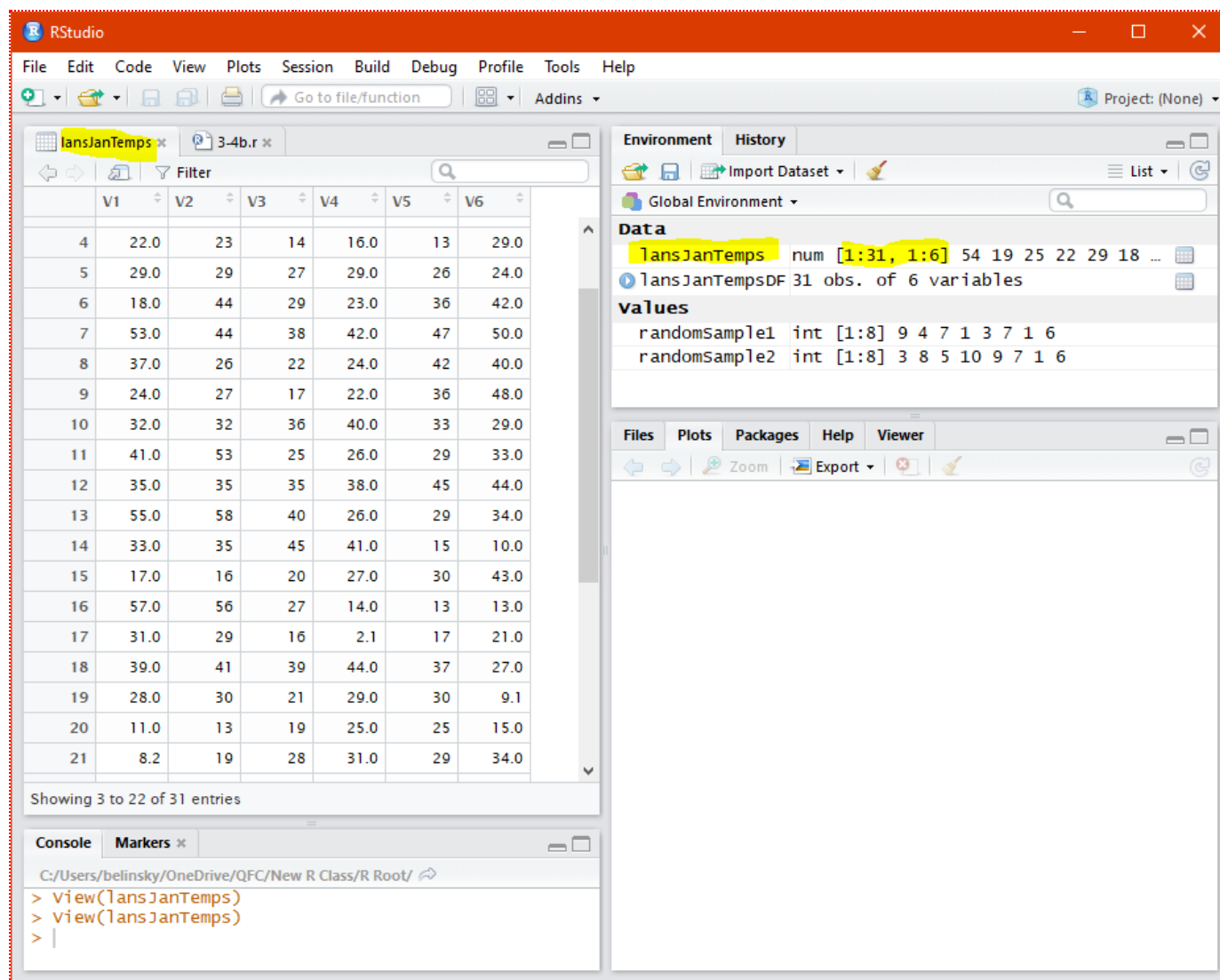


Fig 2: The **lansJanTemp** matrix with six years of January temperatures.

We will take 80 random samples from **lansJanTemps** allowing for repeats and create a histogram of the random observations. We will also add a vertical line to the histogram that represents the mean value.

```
1 randomTemps = sample(x=lansJanTemps, size=80, replace=TRUE);
2 hist(x=randomTemps);
3 abline(v=mean(randomTemps), col="red", lty=4);
```

Note: all 80 values in **randomTemps** can be displayed in the Console Window by typing "lansJanTemps " and hitting Enter.

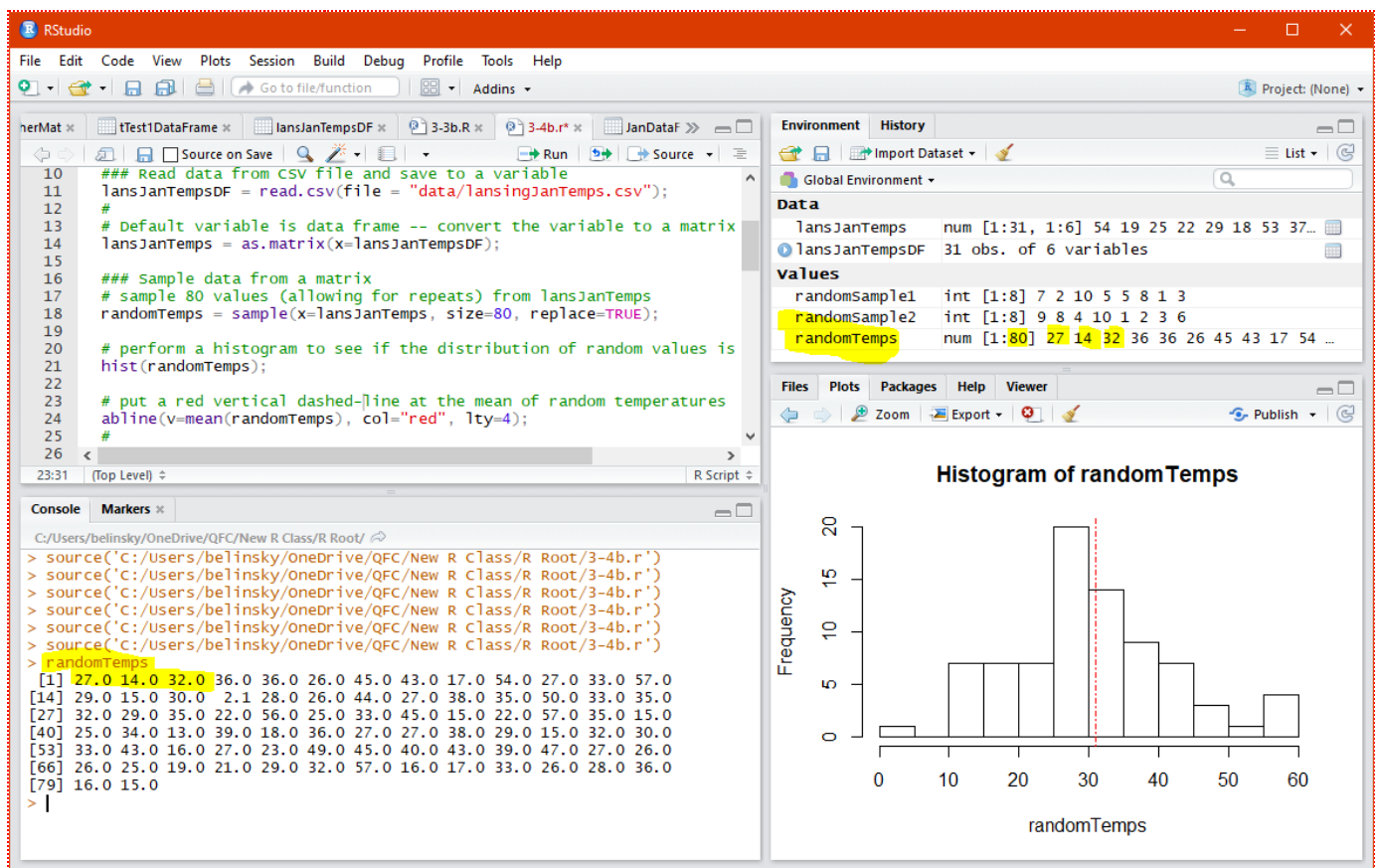


Fig 3: Random sampling of temperature values from the `lansJanTemps` matrix

5 - Sampling from a normal distribution

`sample()` creates a random sample where each observation has the same chance of being picked (i.e., uniform probability). If we want to create a set of observations that comes from a normal distribution with a specified mean and variance we use the function `rnorm()`.

`rnorm()` has 3 parameters:

- **n**: the number of observations to sample
- **mean**: the mean of the distribution
- **sd**: the standard deviation of the distribution

The following code samples and summarizes **200** observations from a normal distribution with a mean of **20** and a standard deviation of **4**.

```
1 normalDist = rnorm(n=200, mean=20, sd=4);
```

The values in `normalDist` can be display visually in a histogram using `hist()`. I have also put a vertical line in to represent the mean value of the `normalDist` using `abline()`:

```
1 hist(x=normalDist, col=c("blue", "green"));
2 abline(v=mean(normalDist), col="red", lty=4, lwd=3); # line type, line width
```

The histogram (Fig.4) shows a fairly normal distribution of values and the mean is close to 20. Of course, the more samples you use, the more likely the distribution will look normal and the mean will be at 20.

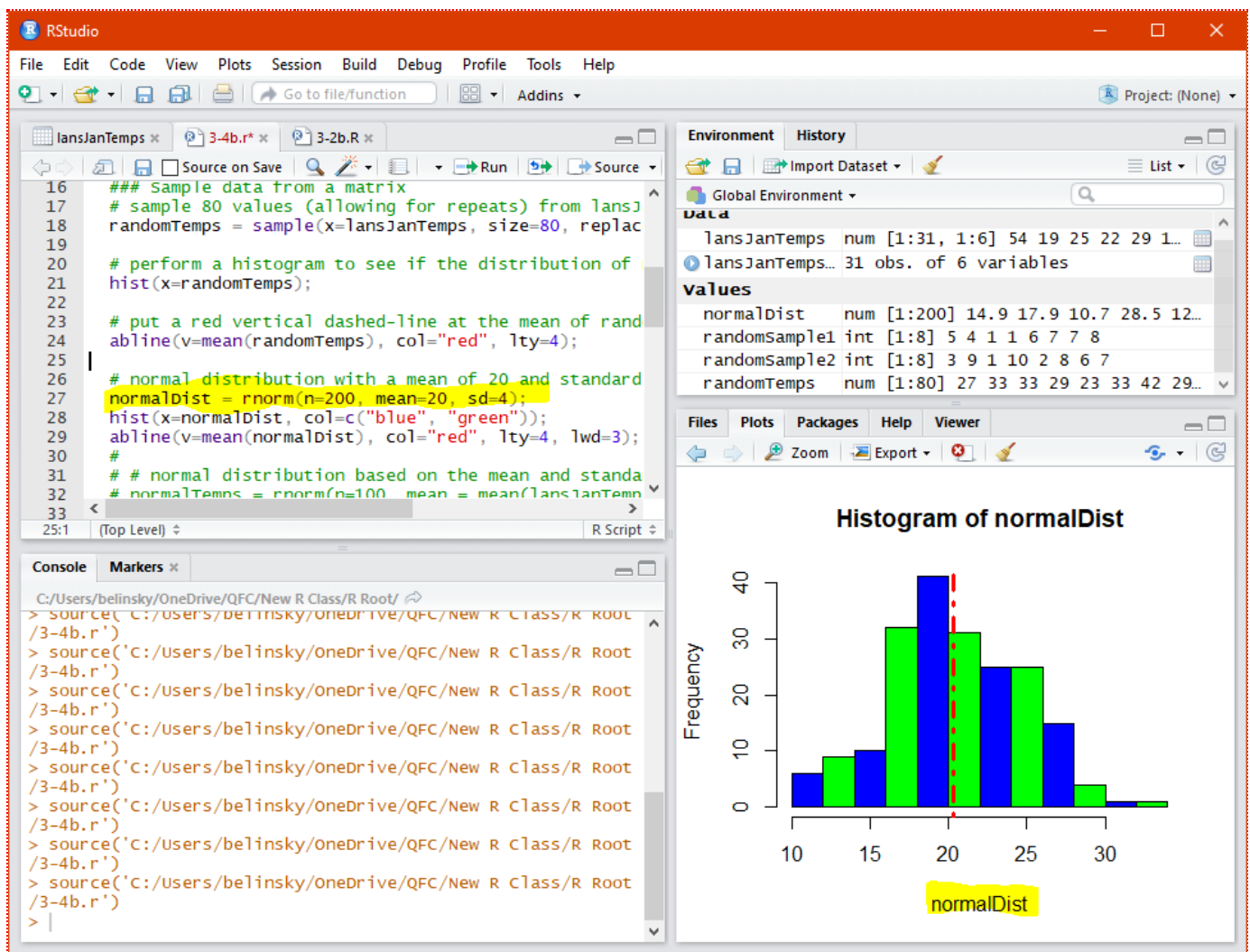


Fig 4: Normal distribution: **200** samples with mean **20** and standard deviation **4**

Note: the values are being picked at random so the histogram will look different each time you execute the script.

5.1 - Normal temperature values

Instead of using hardcoded values (**mean = 20, sd = 4**) for a normal distribution, we are going to sample from a normal distribution with the same mean and standard deviation as the **lansJanTemps** matrix.

We use **mean()** and **sd()** to get the mean and standard deviation from the **lansJanTemps** matrix and use these values as the sample mean and standard deviation in **rnorm()**:

```
1 normalTemps = rnorm(n=100, mean = mean(lansJanTemps),
2                      sd=sd(lansJanTemps));
```

So, in the code above, we generate 100 temperature observations from a normal distribution based on the mean and standard deviation of *lansJanTemps*. The data generated are saved in *normalTemps*.

Then we use a histogram to make a visual representation of **normalTemp** (using **hist()**) and, to the histogram, we add a vertical line at the mean value (using **abline()**):

```
1 hist(normalTemps, angle=60, density=10,  
2       col=c("purple" "black" "red"))
```

```

2   col=c("purple", "black", "red"),
3   main="Histogram of Normal Temps Vector",
4   xlab="Normal Temps");
5 # lty: line type, lwd: line width
6 abline(v=mean(normalTemps), col="darkgreen", lty=6, lwd=3);

```

We can display all 100 observations in the Console Window by typing in "normalTemps" and hitting Enter. The distribution of the 100 temperature observations is shown in the histogram.

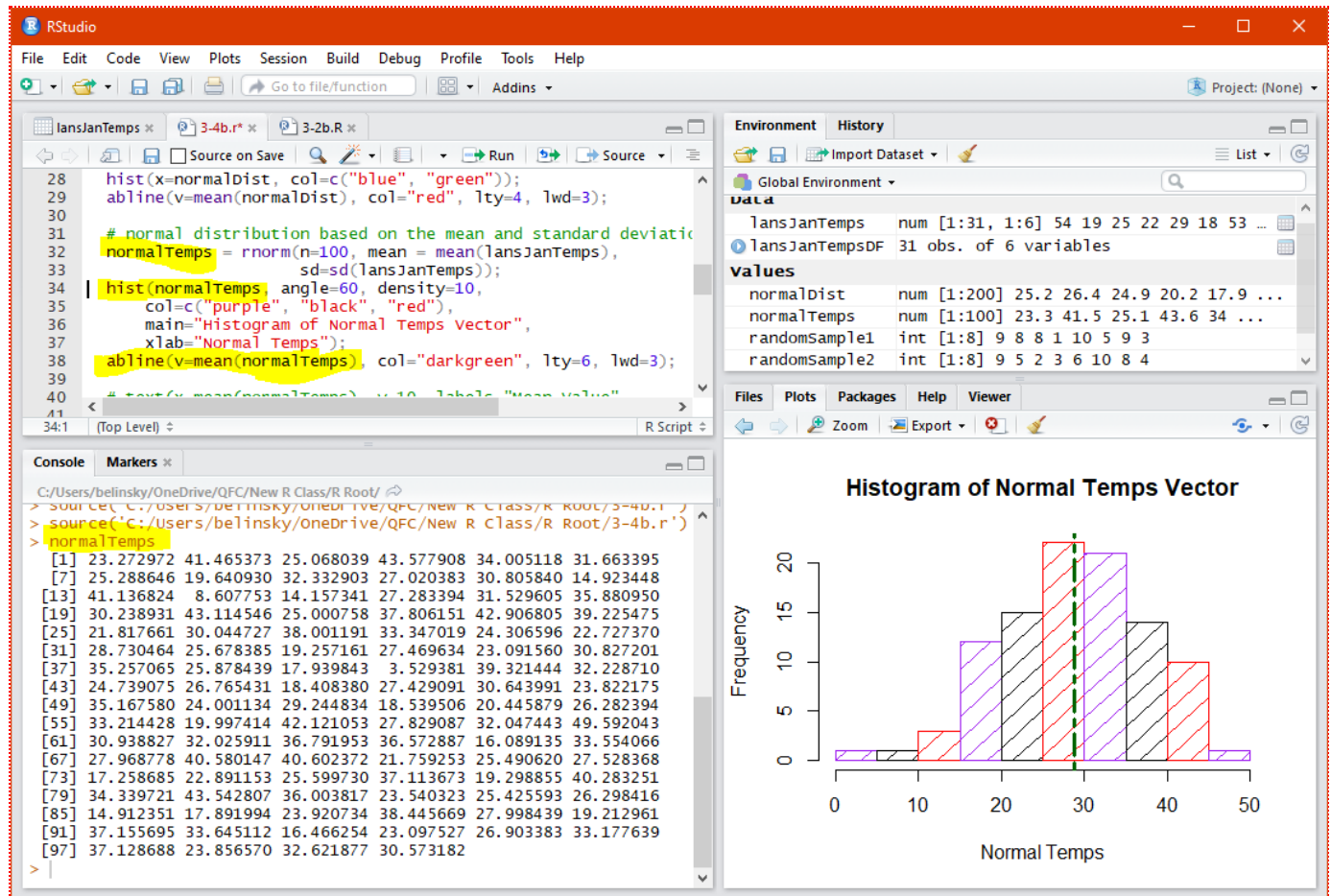


Fig 5: Generating normalized temperature data based on the temperature values from the `lansJanTemps` vector

6 - Psuedo-Random values

Every time your script is executed, a new random set of observations is produced. If you want to replicate your results there is a function called `set.seed()`. `set.seed()` will produce the same "random" numbers every time using a seed number. In other words, `set.seed()` produces observations that look random but the values do not change each time the script is executed.

6.1 - Seed value

Here is a really good [video on PBS Infinite Series](#) about random number generators, pseudo-random number generators, and seed numbers. [The discussion on seed numbers starts at 2:15.](#)

set.seed() takes one parameter and that is the seed number. The seed number is the starting point for a complex formula that produces a series of random-looking, or pseudo-random, values. Here we set the seed number to **23** and ran a couple samples. Each time you execute the code, **seedSample1** and **seedSample2** will have the same value (although they are different from each-other).

```
1 set.seed(seed=23);
2 seedSample1 = sample( x=c(1:100), size=20 );
3 seedSample2 = sample( x=c(1:100), size=20 );
```

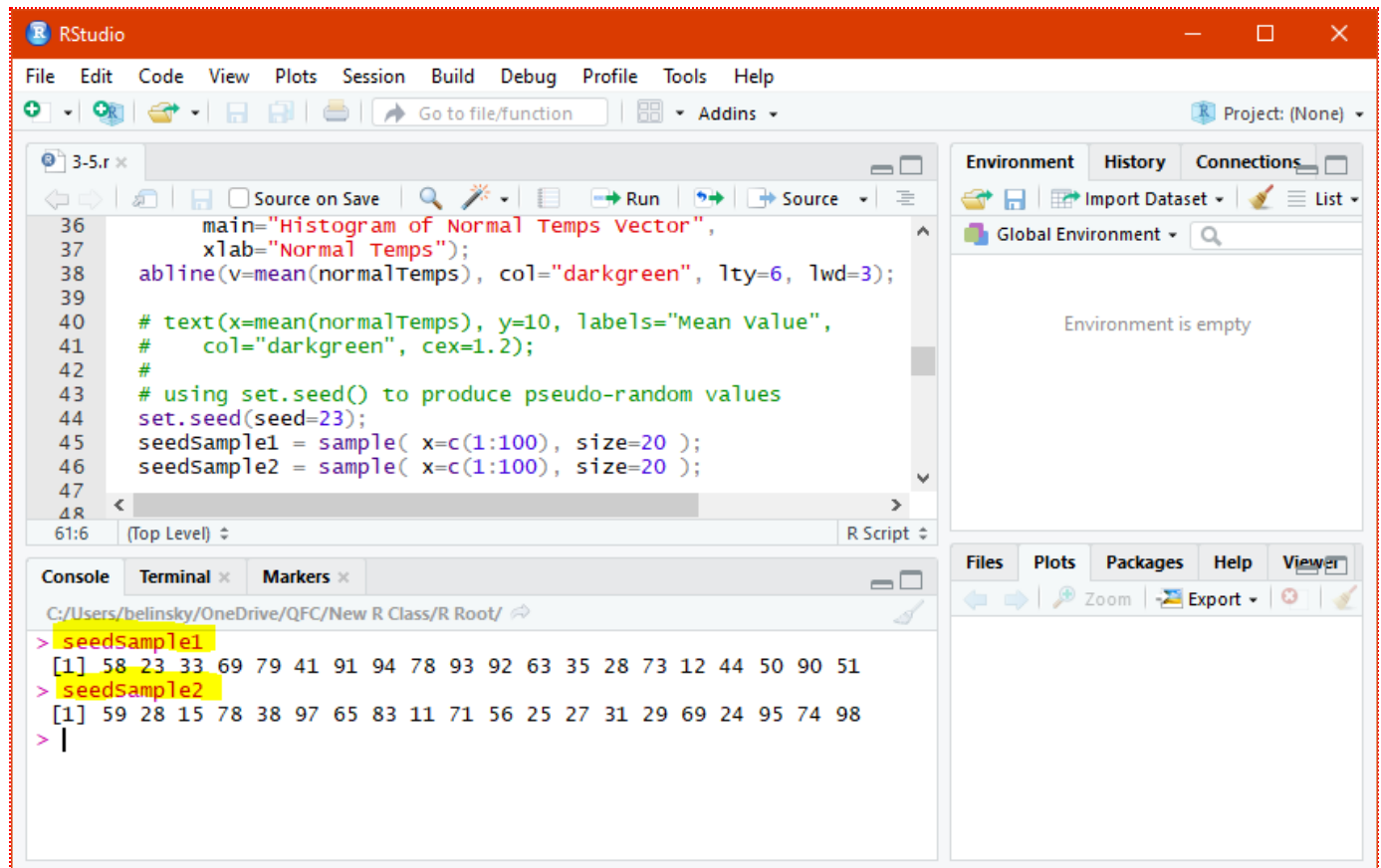


Fig 6: Pseudo-random values produced by a seed number.

6.2 - Changing the seed value

Each time we change the seed value, we "reset" the pseudo-random number generator. **seedSample3** and **seedSample4** will always have the same 20 values based on a generator seeded with the value **28**. **seedSample5** and **seedSample6** will be identical to **seedSample1** and **seedSample2** because the seed value was reset to **23**.

```
1 set.seed(seed=28);

2 seedSample3 = sample( x=c(1:100), size=20 );
3 seedSample4 = sample( x=c(1:100), size=20 );
4
5 set.seed(seed=23); # same seed value as above
6 seedSample5 = sample( x=c(1:100), size=20 ); # will be same as seedSample1
7 seedSample6 = sample( x=c(1:100), size=20 ); # will be same as seedSample2
```

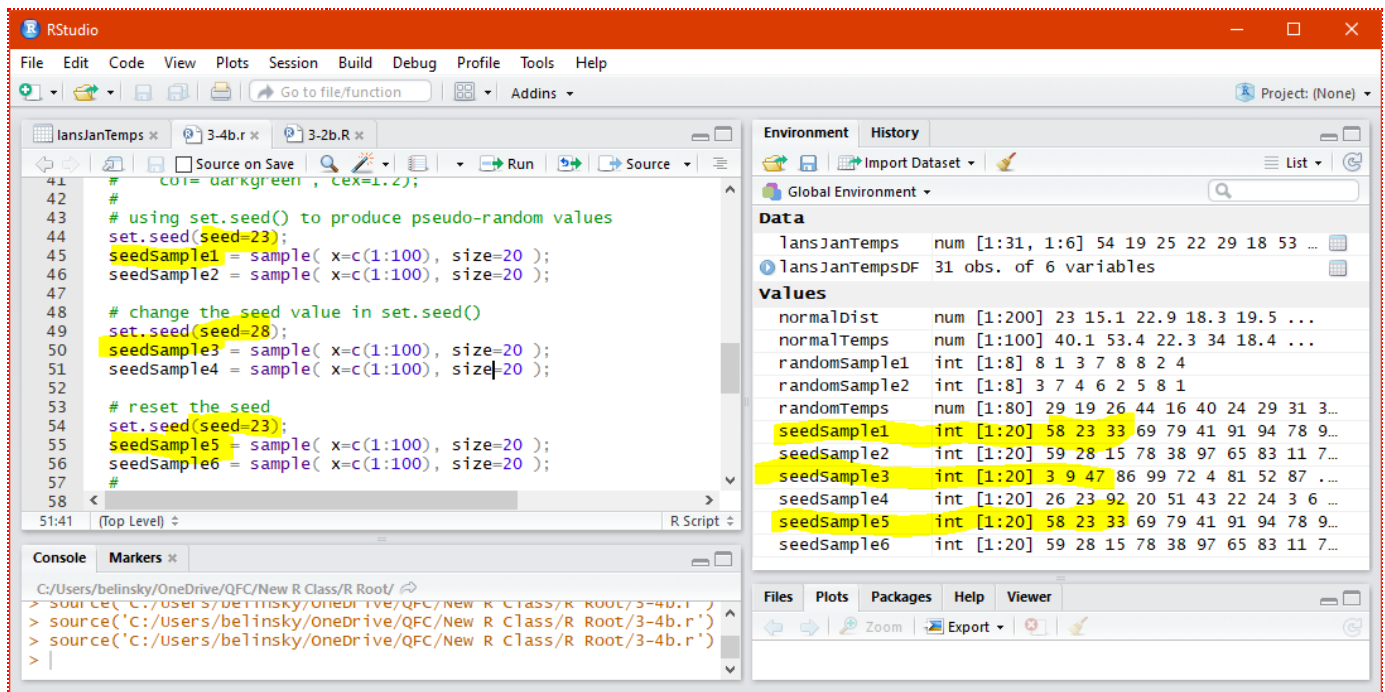


Fig 7: Using `set.seed()` to create pseudo-random values

Note: if you want to stop using the pseudo-random generator in your script, then call:

```
1 set.seed(NULL); # not in the lesson's script
```

6.3 - Pseudo-random values from a matrix

`set.seed()` has been executed in our script and a seed value has been established. So any time we ask for random values in the script, pseudo-random values will be generated based on the seed value (in this case, 23).

The following code *will produce the same* 80 pseudo-random values from the **lansJanTemps** matrix each time:

```
1 setSeedTemps = sample(x=lansJanTemps, size=80, replace=TRUE);
```

And the following code *will produce the same* 100 normalized pseudo-random values based on the mean and standard deviation from **lansJanTemps**.

```
1 setSeedNormalTemps = rnorm(n=100, mean = mean(lansJanTemps),
2                             sd=sd(lansJanTemps));
```

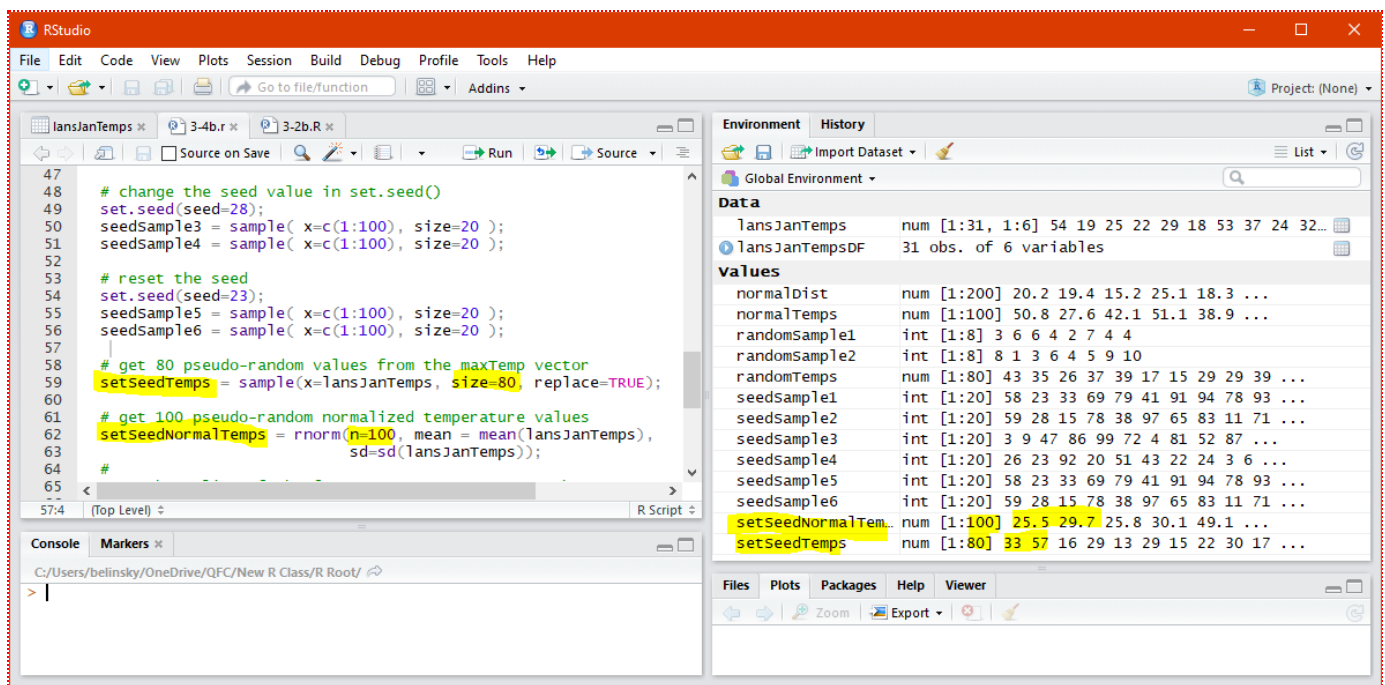



Fig 8: Using `set.seed()` to create pseudo-random temperature data.

Try this out by running the code multiple times without changing the seed number. Then change the seed number and see what happens.

7 - Saving the data you generated

We have generated four different vectors with temperature data in this lesson: ***randomTemps***, ***normalTemps***, ***setSeedTemps***, and ***setSeedNormalTemp***. We might want to save these vectors for later use -- perhaps in another script. However, we cannot save the four vectors to a data frame because the vectors have different sizes (i.e., they have a different number of values). If we want to put the four vectors into one structure, we need to use a ***list***. We can create a list using the ***list()*** function.

```

1 listOfTemps = list("Random_Temps" = randomTemps,
2                     "Normal_Temps" = normalTemps,
3                     "Set_Seed_Temps" = setSeedTemps,
4                     "Set_Seed_Normal_Temps" = setSeedNormalTemps);

```

In this case, the quoted values become the vector's name in the list.

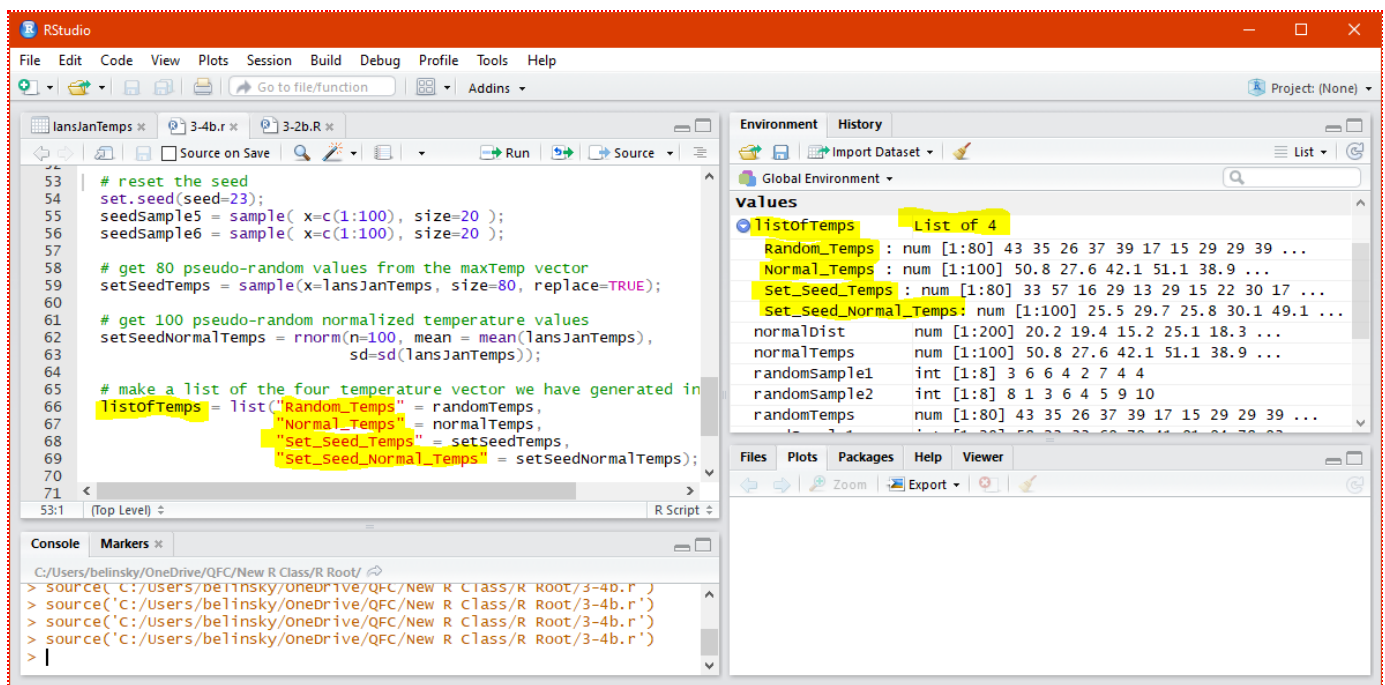


Fig 9: Creating a **list** of the four temperature vectors we created in the script.

7.1 - Saving and loading a list object

We will save the list to a file, called **weatherList.rdata**, so we can access the values from another script.

```
1 save(listOfTemps, file = "data/weatherList.rdata");
```

The file **weatherList.rdata** now exists in your data folder and can be called from another script.

7.2 - Opening the list from another script

From another script, we can load the data from the file **weatherList.rdata**, which resides in your R data directory.

Note: the code in the rest of this lesson is not in the lesson's script.

We use **load()** to load the data from the **weatherList.rdata** file for use in the current script.

```
1 load("data/weatherList.rdata");
```

Notice that this loaded **listOfTemps** that was saved from the other script -- and all the values are the same as before.

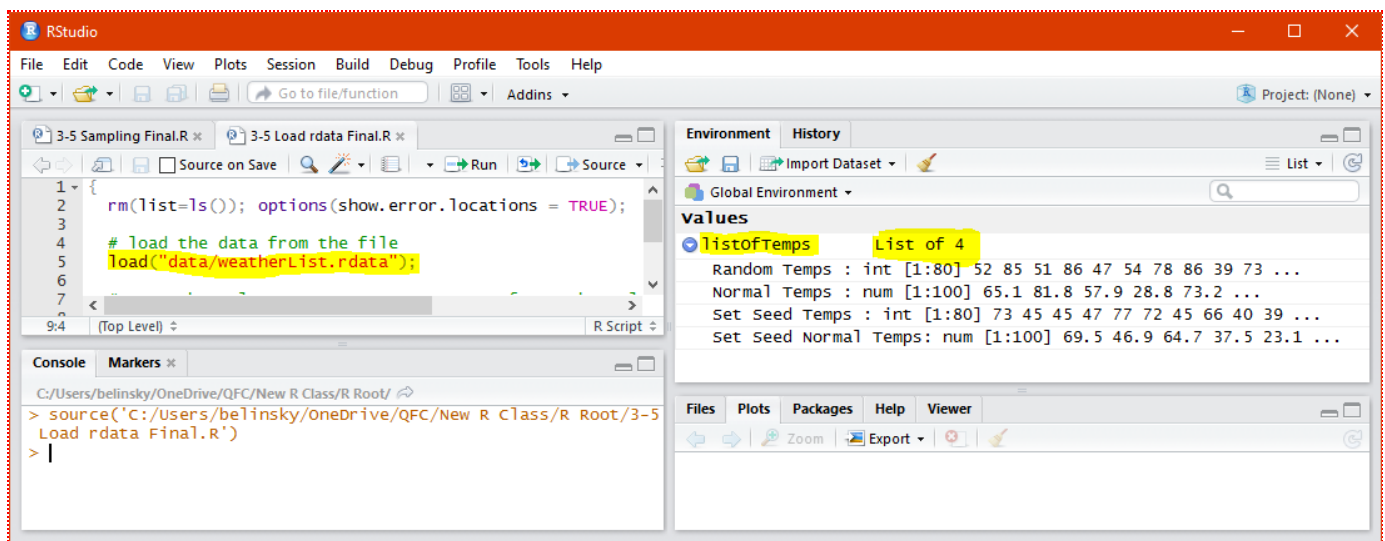


Fig 10: Loading the **list** that was saved from the another script file.

If we want to take elements out of the list and save them to a vector we need to use double brackets `[[]]` instead of single brackets `[]`.

We can refer to the column by its name:

```
1 normalTemps = listOfTemps[["Normal Temps"]];
```

Or by the column number

```
1 setSeedTemps = listOfTemps[[3]];
```

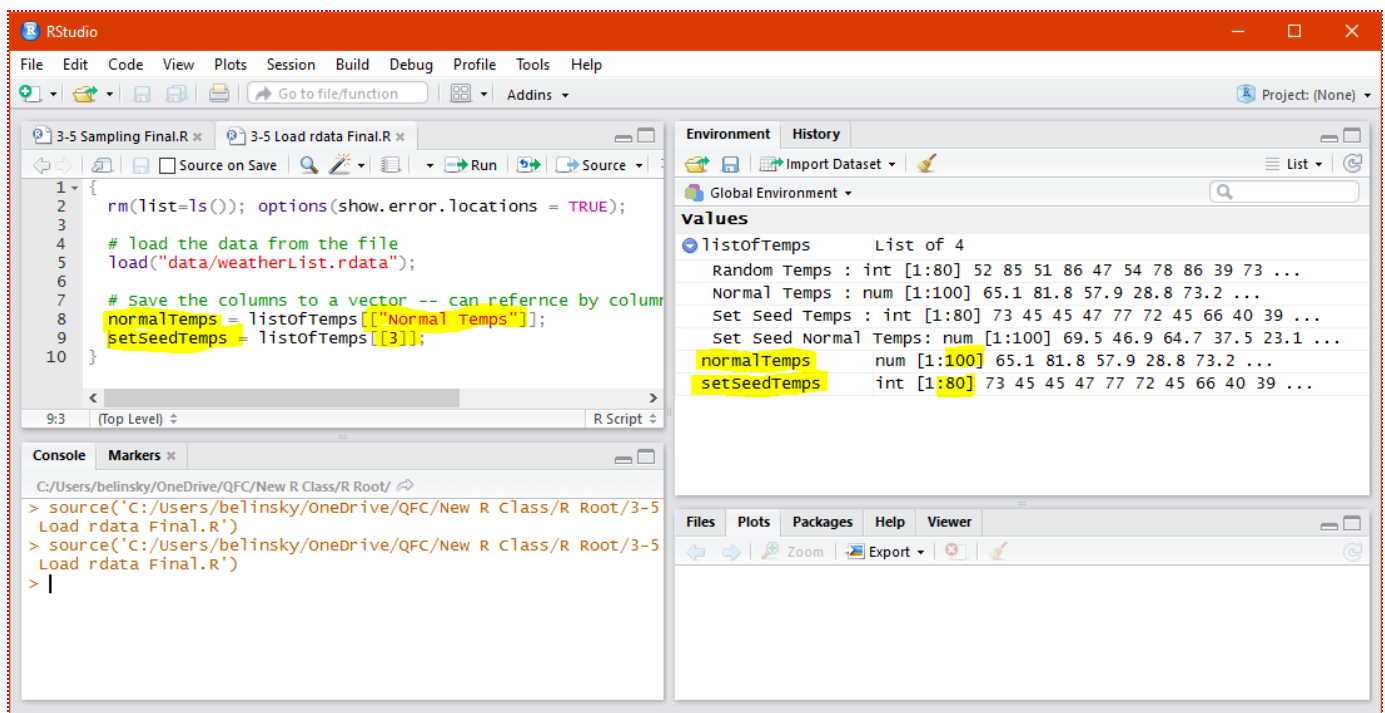


Fig 11: Saving columns from a **list** to a vector.

8 - 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) Using the data from **data/formattedLansingWeather.csv** (created in lesson 3-2, if you don't have the file you can [download it here](#))

- get 50 samples of the average wind speed (AWND) and allow for repeats
- get 30 samples of precipitation (PRCP) and do not allow for repeats

2) Create a set of wind speed values that has a normal distribution with mean and standard deviation equal to the wind speed data from lesson 3-1. Create distributions with $n=20$, 50, and 200 values. Create a histogram for each of the distributions and show where the mean value is.

3) Repeat #2 using a seed value of 15.

Challenge: use a function to do parts two and three. The function would take in a vector (e.g., wind speed), a sample size, and a seed value (**NULL** for no seed value). The function would create the distribution and the histogram from the input values.

*Save your script file as **app3-5.r** in the **scripts** folder of your RStudio Project for the class.*