

# 2-12: Lists

---

## 0.1 To-do

---

## 1 Purpose

---

- Creating you own List
  - Subsetting a List
  - Reading a List return from plots and models
- 

## 2 Script for this lesson

---

The [script for the lesson is here](#)

The [data used for this lesson and the application](#) is here

---

## 3 Objects in R

---

As mentioned in last lesson, there are 2 types of objects (i.e., variables) in R:

- Atomic vector (often just called vectors): objects that hold values of the same type (e.g., numeric, string, logical)
- Lists: objects that hold other objects (including more Lists)

The relationship between Atomic Vectors and Lists is analogous to the relationship between folders and files on your computer. Files, like atomic vector, hold information, or data. Folders, like Lists, are containers for other folders and files.

Last lesson we talked mostly about atomic vectors. This lesson we will talk mostly about Lists. As you will see in this lesson, Lists create a tree-like structure for your data. At the end of the tree is the Atomic Vectors

---

## 4 Lists

---

A List is an object that holds other (usually related) objects. When you create a plot or perform a statistical test, the results are often stored as Lists, as we will see later in the lesson.

We will create our own List with three objects in it.

The three objects will be:

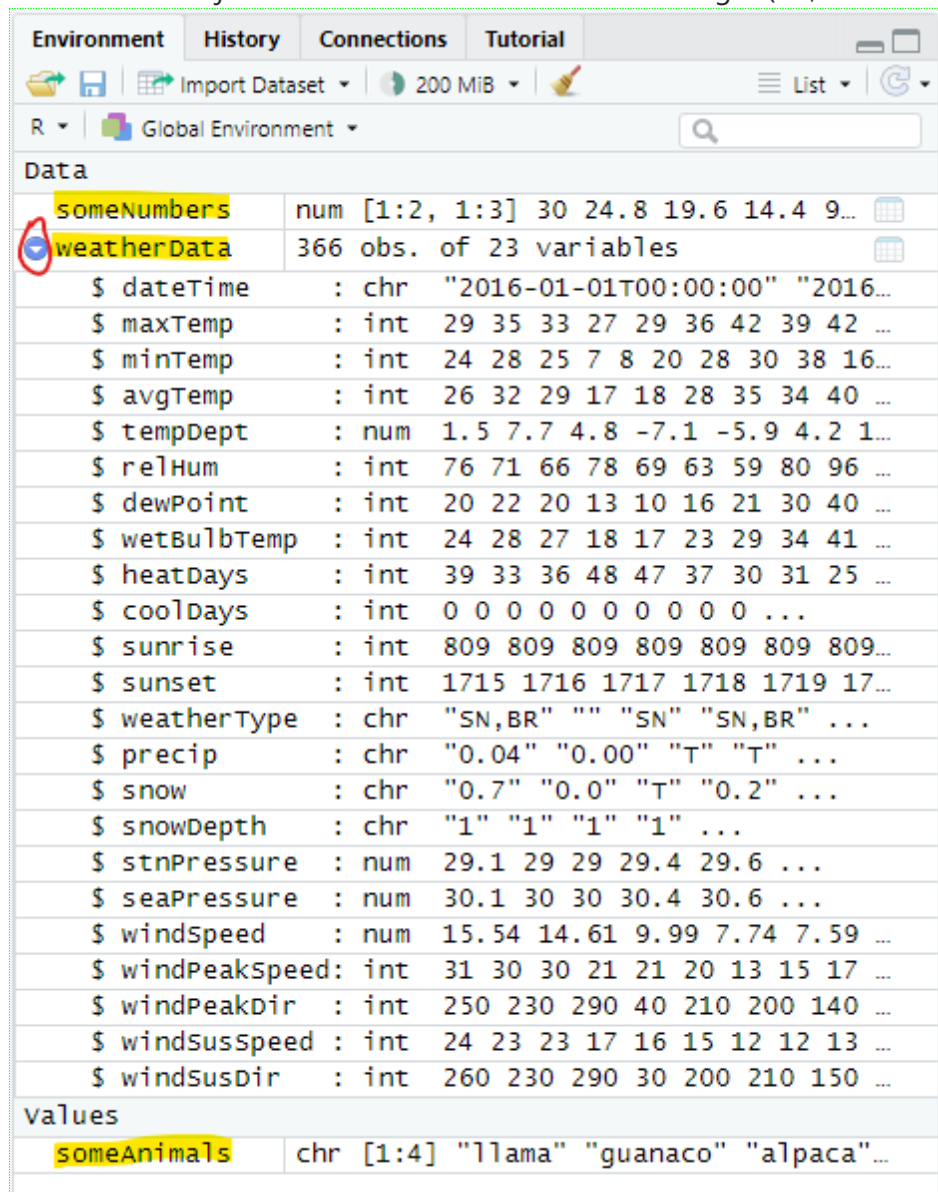
- a character vector called ***someAnimals***

- a 2x3 matrix consisting of a sequence of six numbers called **someNumbers**
- the **weatherData** dataframe from previous lessons

```
someAnimals = c("llama", "guanaco", "alpaca", "goat");
someNumbers = matrix(nrow=2, ncol=3, seq(from=30, to=4, length.out=6))
weatherData = read.csv(file="data/Lansing2016NOAA.csv");
```

## 4.1 Lists in the Environment tab

Notice that, in the Environment, **weatherData** has an arrow to expand it whereas **someAnimals** and **someNumbers** contains all of their information in one line. It is this expand arrow that indicates **weatherData** is a List, whereas **someAnimals** and **SomeNumbers** are atomic vectors. **weatherData** is a specialized List where all the objects inside are vectors of the same length (i.e., **weatherData** is a data frame):



Environment	History	Connections	Tutorial
R   Global Environment			
Data			
someNumbers	num [1:2, 1:3] 30 24.8 19.6 14.4 9...		
weatherData	366 obs. of 23 variables		
\$ dateTime	: chr	"2016-01-01T00:00:00" "2016...	
\$ maxTemp	: int	29 35 33 27 29 36 42 39 42 ...	
\$ minTemp	: int	24 28 25 7 8 20 28 30 38 16...	
\$ avgTemp	: int	26 32 29 17 18 28 35 34 40 ...	
\$ tempDept	: num	1.5 7.7 4.8 -7.1 -5.9 4.2 1...	
\$ relHum	: int	76 71 66 78 69 63 59 80 96 ...	
\$ dewPoint	: int	20 22 20 13 10 16 21 30 40 ...	
\$ wetBulbTemp	: int	24 28 27 18 17 23 29 34 41 ...	
\$ heatDays	: int	39 33 36 48 47 37 30 31 25 ...	
\$ coolDays	: int	0 0 0 0 0 0 0 0 0 ...	
\$ sunrise	: int	809 809 809 809 809 809 809...	
\$ sunset	: int	1715 1716 1717 1718 1719 17...	
\$ weatherType	: chr	"SN,BR" "" "SN" "SN,BR" ...	
\$ precip	: chr	"0.04" "0.00" "T" "T" ...	
\$ snow	: chr	"0.7" "0.0" "T" "0.2" ...	
\$ snowDepth	: chr	"1" "1" "1" "1" ...	
\$ stnPressure	: num	29.1 29 29 29.4 29.6 ...	
\$ seaPressure	: num	30.1 30 30 30.4 30.6 ...	
\$ windspeed	: num	15.54 14.61 9.99 7.74 7.59 ...	
\$ windPeakspeed	: int	31 30 30 21 21 20 13 15 17 ...	
\$ windPeakDir	: int	250 230 290 40 210 200 140 ...	
\$ windsusspeed	: int	24 23 23 17 16 15 12 12 13 ...	
\$ windsusDir	: int	260 230 290 30 200 210 150 ...	
values			
someAnimals	chr [1:4]	"llama" "guanaco" "alpaca"...	

Figure 1: The expand arrow in the Environment indicates that the object is a List (i.e., holds other objects)

## 4.2 Creating the new List

We can create a List with the three objects above using the **list()** function:

```
listAtOnce = list(someAnimals, someNumbers, weatherData);
```

This will add **listAtOnce** to the **Environment**. From there we double-click on it to see it in a **Viewer** tab. The **Viewer** tab offers a more helpful and informative view of the List.

A couple things to notice about **listAtOnce** in the **Viewer** tab:

- The objects inside the List do not have names – instead they only have numbers. **Objects do not retain their names when put in to a List.**
- The third object **[[3]]**, the dataframe, has an expand arrow. The arrow indicates that this object is also a List – **[[1]]** and **[[2]]** are atomic vectors.

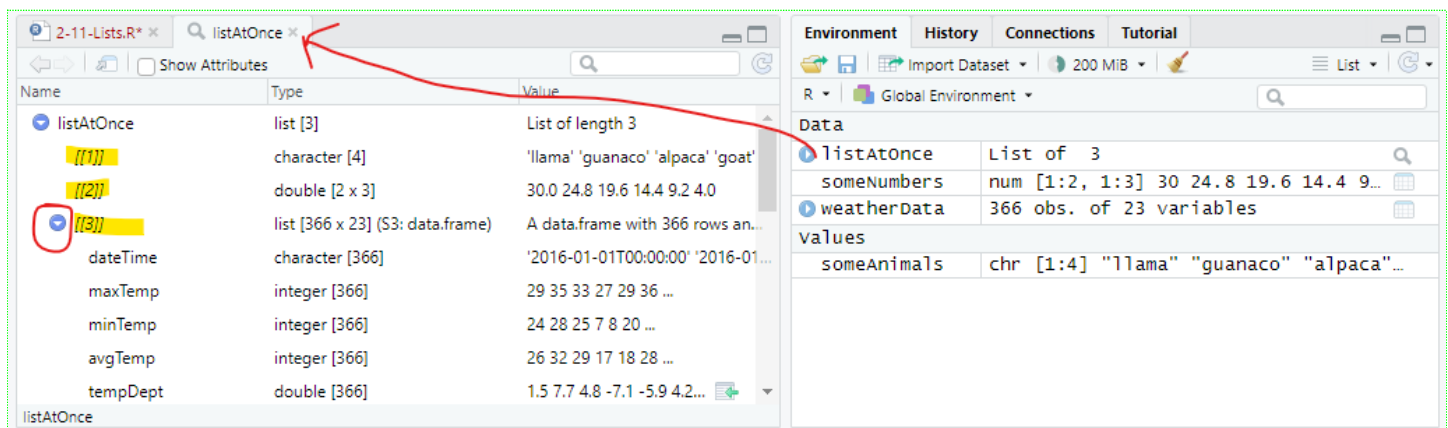


Figure 2: The List inside a Viewer tab

Note: When looking at the List in the Viewer, the objects with expand arrows are functionally similar to folders, those without arrows are functionally similar to files

## 4.3 Including names for the objects

If we want the objects inside the List to have names, then we have to specify the names while creating the List. You do that by setting a name equal to the object when calling **list()**:

```
listAtOnce2 = list(animals = someAnimals,  
                  numbers = someNumbers,  
                  weather = weatherData);
```

This makes **animals**, **numbers**, and **weather** the names of the objects **someAnimal**, **someNumbers**, and **weatherData** inside the List. You can choose any name, including the same as the original objects:

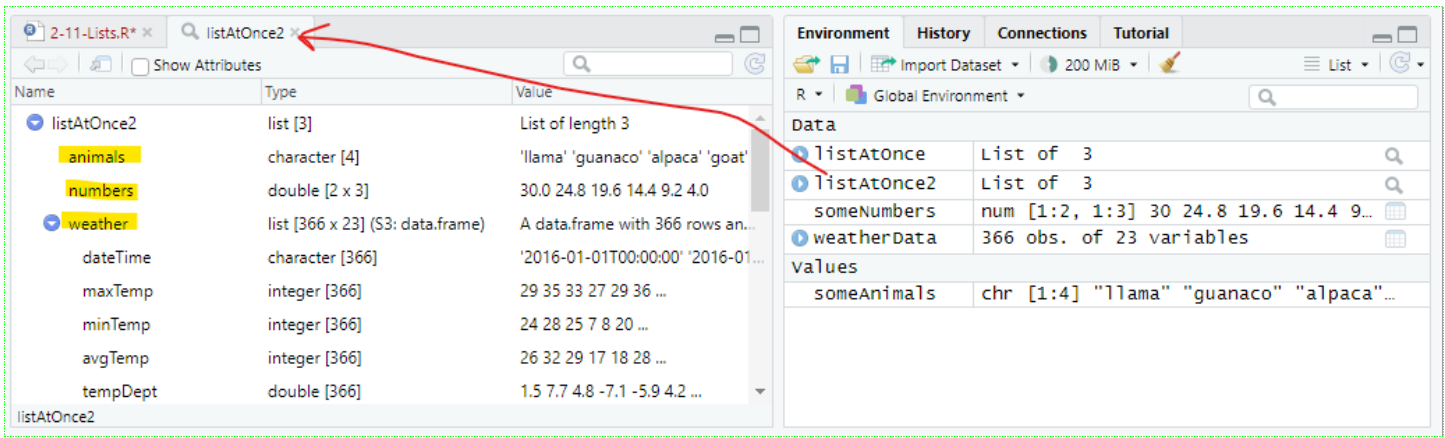


Figure 3: Adding names to the objects inside the List

Note: names inside Lists, like variables, should follow programming naming conventions – but this is not enforced in R.

## 5 Creating a dynamic List

In the previous example, all the objects put in the List were put in at once. What often happens is that data is generated and gets dynamically added to a List, so the objects are put in the List separately. We are going to create a List equivalent to **listAtOnce** except the three object will be put in one at a time.

First we create a List with nothing in it (i.e., an empty list):

```
listDynamic = list();
```

Environment

```
listDynamic: List of 0
```

### 5.1 Appending to the List

Then add the object **someAnimals** to the List.

```
listDynamic2 = append(listDynamic, someAnimals);
```

Somewhat unintuitively, **append()** does not append the object **someAnimals** to the List. Instead, **append()** appends each value within **someAnimals** to the List – making each value a separate object. Now you have a List with 4 string vectors (each with 1 value) .... this is not what we want.

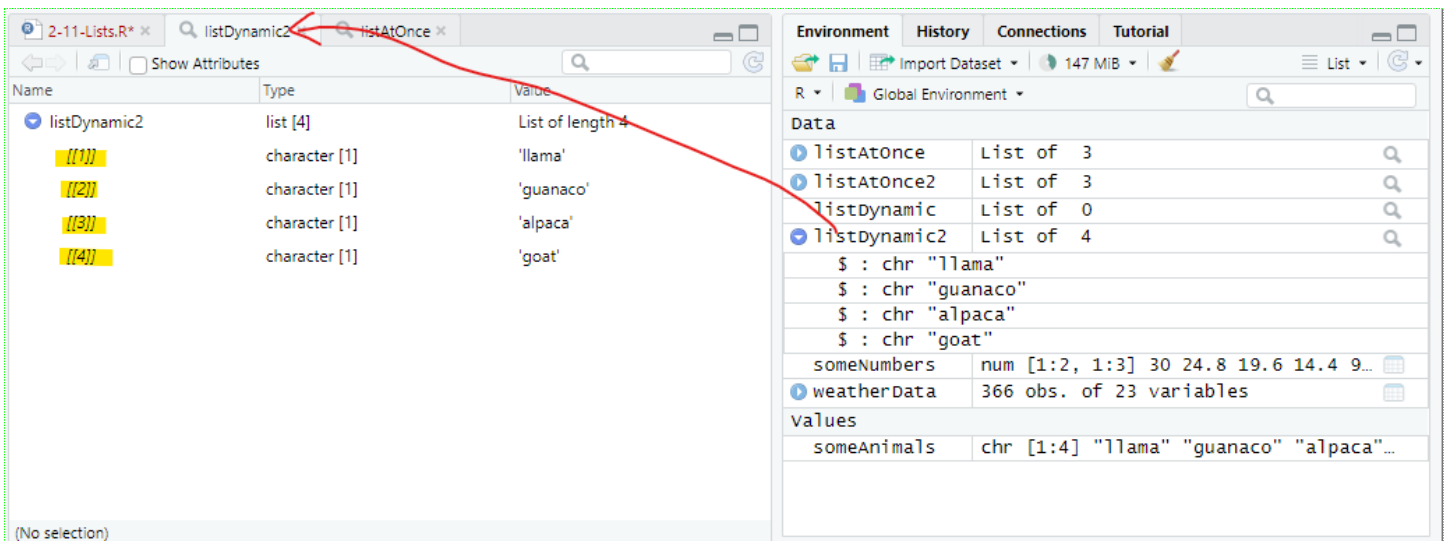


Figure 4: Appending an object one value at a time...

## 5.2 Appending whole objects

The solution is also a bit unintuitive. We want is to append **someAnimals** to the List as vector and, to do this, we need to wrap **someAnimals** inside its own List:

```
listDynamic3 = append(listDynamic, list(someAnimals));
```

And, if we want the object to have a name inside the List, then we need to provide the name – just like [Figure 3](#) :

```
listDynamic4 = append(listDynamic, list(animal = someAnimals));
```

**listDynamic3** and **listDynamic4** both now have a vector with four character values, the latter has that vector named:

Environment		History	Connections	Tutorial
<div> <div>Import Dataset</div> <div>149 MiB</div> <div>List</div> </div>				
R Global Environment				
Data				
listAtOnce	List of 3			
listAtOnce2	List of 3			
listDynamic	List of 0			
listDynamic2	List of 4			
listDynamic3	List of 1			
\$ : chr [1:4] "llama" "guanaco" "alpaca" "goat"				
listDynamic4	List of 1			
\$ animal: chr [1:4] "llama" "guanaco" "alpaca" "goat"				
someNumbers	num [1:2, 1:3] 30 24.8 19.6 14.4 9.2 4			
weatherData	366 obs. of 23 variables			
Values				
someAnimals	chr [1:4] "llama" "guanaco" "alpaca" "goat"			

Figure 5: Lists with appended objects

### 5.3 Appending multiple objects

We will now append **someNumber** and **weatherData** using **append()**:

```
listDynamic5 = append(listDynamic4, list(numbers=someNumbers));
listDynamic5 = append(listDynamic5, list(weather=weatherData));
```

And see that **listDynamic5** is a List with three objects in it: **animals**, **numbers**, and **weather** – just like **listAtOnce2** (Figure 3).

Environment		History	Connections	Tutorial
<div> <div>Import Dataset</div> <div>149 MiB</div> <div>List</div> </div>				
R Global Environment				
Data				
listDynamic4	List of 1			
listDynamic5	List of 3			
\$ animal: chr [1:4] "llama" "guanaco" "alpaca" "goat"				
\$ numbers: num [1:2, 1:3] 30 24.8 19.6 14.4 9.2 4				
\$ weather: 'data.frame': 366 obs. of 23 variable...				
.. \$ dateTime : chr [1:366] "2016-01-01T00:00:00" "..."				
.. \$ maxTemp : int [1:366] 29 35 33 27 29 36 42 39...				
.. \$ minTemp : int [1:366] 24 28 25 7 8 20 28 30 3...				
.. \$ avgTemp : int [1:366] 26 32 29 17 18 28 35 34...				
.. \$ tempDept : num [1:366] 1.5 7.7 4.8 -7.1 -5.9 4...				
.. \$ relHum : int [1:366] 76 71 66 78 69 63 59 80...				
.. \$ dewPoint : int [1:366] 20 22 20 13 10 16 21 30...				
.. \$ wetBulbTemp : int [1:366] 24 28 27 18 17 23 29 34...				
.. \$ heatDays : int [1:366] 39 33 36 48 47 37 30 31...				
.. \$ coolDays : int [1:366] 0 0 0 0 0 0 0 0 0 ...				
.. \$ sunrise : int [1:366] 809 809 809 809 809 809...				
.. \$ sunset : int [1:366] 1715 1716 1717 1718 171...				

Figure 6: Lists after the three objects were appended and named

## 6 Subsetting within a List

The RStudio **Viewer** tab for Lists has a nice feature where it will show you how to subset an object inside a List when you click on it. In **listDynamic5** List, when you click on **avgTemp** in the **weather** dataframe – and at the bottom of the tab, RStudio shows the code to subset **avgTemp**:

```
listDynamic5[["weather"]][["avgTemp"]]
```

If you click the document icon with an arrow on the right-side, RStudio will put the above code in the **Console** where you click **Enter** to execute it:

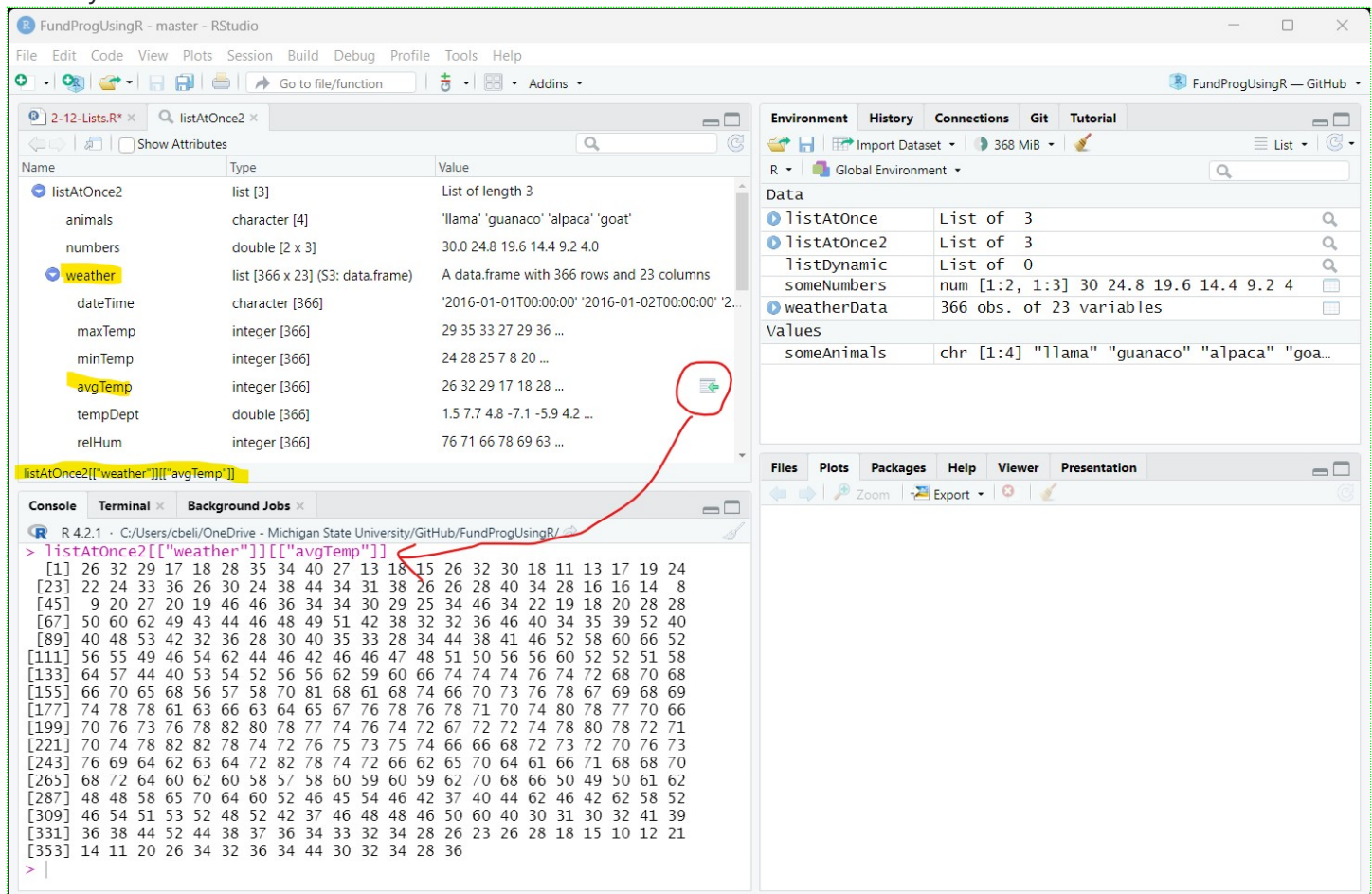


Figure 7: Opening an object within a List from the Viewer tab

### 6.1 Subsetting with \$ and [[ ]]

Every object in the List Viewer with a value under **Name** in Figure 7 can be accessed with both the **\$** and **[[ ]]** subset operators – RStudio always uses **[[ ]]**.

Here are two ways to subset the **animal** object inside **listDynamic5** – the results are saved to **anim1** and **anim2**:



```
anim1 = listDynamic5[["animal"]];  
anim2 = listDynamic5$animal;
```

The **string** (chr) vectors **anim1** and **anim2** have the same values as the original **someAnimals** vector:

#### Environment

```
anim1:      chr [1:4] "llama" "guanaco" ...  
anim2:      chr [1:4] "llama" "guanaco" ...  
someAnimals: chr [1:4] "llama" "guanaco" ...
```

Here are three ways to subset the **dewPoint** vector within **weather**, which is within **listDynamic5**:

```
dewPoint1 = listDynamic5$weather$dewPoint;  
dewPoint2 = listDynamic5[["weather"]][["dewPoint"]];  
dewPoint3 = listDynamic5[["weather"]]$dewPoint;
```

Again, all three saved values are identical to the original **weatherData\$dewPoint**:

```
dewPoint1:      20 22 20 13 10 16 ...  
dewPoint2:      20 22 20 13 10 16 ...  
dewPoint3:      20 22 20 13 10 16 ...  
▼ weatherData: 366 obs. of 23 variables  
  $ dewPoint:   20 22 20 13 10 16...
```

**\$** and **[[ ]]** are equivalent operators when you are working with named objects within a List. **\$** is more convenient to use because it involves less typing and RStudio will give you suggestions.

---

## 6.2 Numeric subsetting (only with **[[ ]]**)

Objects within a Lists can also be accessed by their numeric order using **[[ ]]**:

```
anim3 = listDynamic5[[1]];  
dewPoint4 = listDynamic5[[3]][[7]];
```

But, you cannot use the **\$** operator to subset by number:

```
# anim3 = listDynamic5$1;    # will cause an error
```

So, if the object inside a List does not have a name, as in **listAtOnce** (Figure 2), then you have to use **[[ ]]** to subset the unnamed object by numeric placement.

---

## 6.3 The **[ ]** subset operator

You can use **[ ]** to subset the List, but instead of getting the object, you will get a List with the object in it:

```
anim4 = listDynamic5["animal"];
```



In the **Environment**, we can see that **anim1**, **anim2**, and **anim3** (subsetting with either `[[ ]]` or `$`) are all character vectors with 4 values. But, **anim4** (subsetting with `[ ]`) is a List with a character vector that has 4 values:

```
Environment

anim1: chr [1:4] "llama" "guanaco" ...
anim2: chr [1:4] "llama" "guanaco" ...
anim3: chr [1:4] "llama" "guanaco" ...

▼ anim4: List of 1
  $ animal: chr [1:4] "llama" "guanaco" ...
```

Figure 8: The difference between using `[[ ]]` or `$` vs `[ ]` for subsetting objects within a List

This is not very useful so `[ ]` is rarely used to subset a List.

---

## 7 Returned List from functions

---

We are going to move from creating our own Lists to looking at Lists that are returned when you call certain functions, specifically **ggplot()** and **lm()**.

First, we will copy the scatterplot and linear regression of humidity vs. temperature from lesson 2-1 and save the results, which will be a List, to **plot1**:

```
plot1 = ggplot( data=weatherData ) +
  geom_point( mapping=aes(x=avgTemp, y=relHum) ) +
  geom_smooth( mapping=aes(x=avgTemp, y=relHum),
              method="lm" ) +
  labs( title="Humidity vs Temperature",
        subtitle="Lansing, MI -- 2016",
        x = "Average Temperatures (Fahrenheit)",
        y = "Relative Humidity") +
  scale_x_continuous( breaks = seq(from=10, to=80, by=10) ) +
  theme_bw() +
  theme( axis.text.x=element_text(angle=90, vjust=0.5) );
plot(plot1);
```

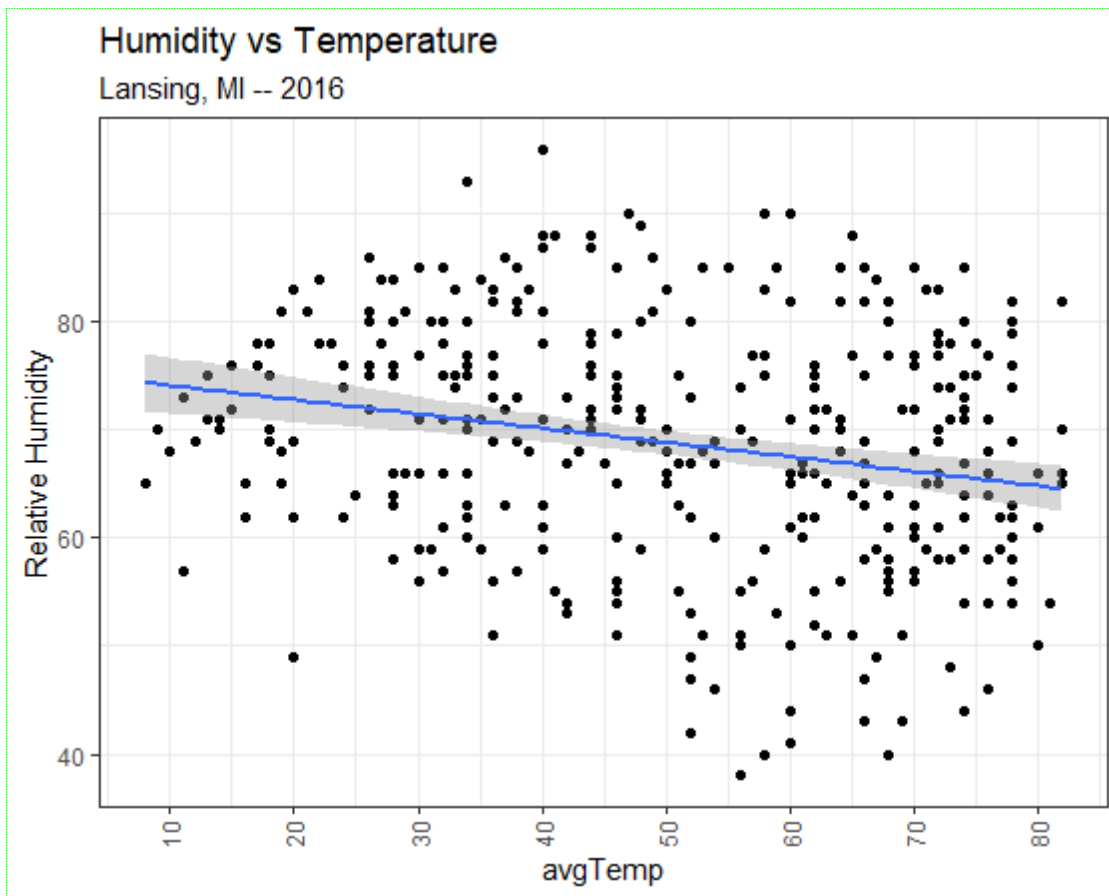


Figure 9: Scatterplot of humidity vs. temperature with a linear regression

We can also create a linear regression model of humidity vs temperature using ***lm()*** and save the results to a list named ***model1***:

```
model1 = lm(formula=weatherData$avgTemp~weatherData$relHum);
```

And then look at a summary of the model using ***print(summary(model1))***:

```
> print(summary(model1));
```

Call:

```
lm(formula = weatherData$avgTemp ~ weatherData$relHum)
```

Residuals:

Min	1Q	Median	3Q	Max
-44.213	-14.424	-0.213	15.479	36.461

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	77.27006	6.07644	12.716	< 2e-16 ***
weatherData\$relHum	-0.38696	0.08722	-4.437	1.21e-05 ***

---

Signif. codes:

```
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 18.59 on 364 degrees of freedom

Multiple R-squared: 0.0513, Adjusted R-squared: 0.04869

F-statistic: 19.68 on 1 and 364 DF, p-value: 1.213e-05

Figure 10: Summary of a model of a linear regression

Both **plot1** and **model1** are List objects and the information for the plot and summary of the linear model are either in the List, or calculated from values in the List.

## 7.1 The returned List: linear model

The plot (Figure 9) and the summary (Figure 10) above are both ways of getting at the data that comes back from the corresponding functions calls **ggplot()** and **lm()**. But, both of these calls also return a List that contains a lot of information.

Let's first look at **model1**, which is the List returned from the **lm()** call:

The screenshot shows the RStudio interface. The top-left pane displays the R console output from the `print(summary(model1))` command. The top-right pane shows the Environment window, which lists objects in the Global Environment. The `model1` object is highlighted in yellow. The bottom-left pane shows the structure of the `model1` list, which contains 12 elements: `coefficients`, `residuals`, `effects`, `rank`, `fitted.values`, `assign`, `qr`, `df.residual`, `xlevels`, `call`, `terms`, and `model`. A red arrow points from the `model1` entry in the Environment pane to the `model1` entry in the top-left pane.

Figure 11: The List returned from a `lm()` call

## 7.2 Subsetting the linear model List

**model1** has information that is inside the summary, for instance the estimate for the intercept is at:

Console

```
> model1$coefficients["(Intercept)"]  
(Intercept)  
77.27006
```

Note: **(Intercept)** does not follow good programming naming practices!

We need to use `[ ]` to subset **coefficients** because **coefficients** is an atomic vector. Note: Subsetting operators are a bit inconsistent in R and it is just something you need to get used to.

We can also subset the first ten **residuals** from **model1**:

Console

```
> model1$residuals[1:10]  
      1      2      3      4      5  
-21.8609230 -17.7957344 -22.7305458 -30.0869984 -32.5696589  
      6      7      8      9     10  
-24.8914326 -19.4392818 -12.3130738 -0.1216773 -20.0869984
```

Or, subset every twentieth value in **fitted.values**:

Console

```
> model1$fitted.values[seq(from=1, to=366, by=20)]  
      1     21     41     61     81    101    121    141  
47.86092 45.92611 52.11751 44.76522 57.53498 46.31307 49.02181 57.92194  
    161    181    201    221    241    261    281    301  
54.43928 57.53498 54.82624 55.21321 47.47396 45.15219 50.56966 43.60434  
    321    341    361  
48.24789 46.31307 46.70004
```

There are also a lot of other, more obscure, values in the List object which we will not go through. The point is that there is a lot of information inside the List and that information can be extracted using subset operators.

---

## 7.3 New variables types

If you look through **model1**, you will find three new variable types: **language**, **symbol**, and **formula**. These variables types are used by R programmers to store the code that created the List – this is not something you will need to deal with at this point.

Name	Type	Value
model1	list [12] (S3: lm)	List of length 12
coefficients	double [2]	77.270 -0.387
residuals	double [366]	-21.9 -17.8 -22.7 -30.1 -32.6 -24.9 ...
effects	double [366]	-969.2 82.5 -21.9 -28.3 -31.5 -24.3 ...
rank	integer [1]	2
fitted.values	double [366]	47.9 49.8 51.7 47.1 50.6 52.9 ...
assign	integer [2]	0 1
qr	list [5] (S3: qr)	List of length 5
df.residual	integer [1]	364
xlevels	list [0]	List of length 0
call	language	lm(formula = weatherData\$avgTemp ~ we...
[[1]]	symbol	`lm`
formula	language	weatherData\$avgTemp ~ weatherData\$rel...
[[1]]	symbol	`~`
[[2]]	language	weatherData\$avgTemp
[[3]]	language	weatherData\$relHum
terms	formula	weatherData\$avgTemp ~ weatherData\$rel...
model	list [366 x 2] (S3: data.frame)	A data.frame with 366 rows and 2 columns

Figure 12: Some of the less common variable types appear in this List

## 7.4 Return list: plot1

The List returned when **ggplot()** is called is far more complex than the one returned with **lm()**. Inside the **ggplot()** List are many embedded Lists that contain everything required to make the plot, including all the data used in the plot and the styling.

There is also a new object type called **environment**, which is sort of a modern version of a List. The name is confusing because it is structurally related to, but not the same thing as, the **Environment** tab in RStudio. This is not something we will explore in this class – just know that an **environment** object, like Lists, holds other objects and is also analogous to folders on your computer.

Name	Type	Value
plot1	list [9] (S3: gg, ggplot)	List of length 9
data	list [366 x 23] (S3: data.frame)	A data.frame with 366 rows and 23 columns
layers	list [2]	List of length 2
scales	environment [2] (S3: ScalesList, ggproto)	<environment: 0x000002477d830ef8>
mapping	list [0] (S3: uneval)	List of length 0
theme	list [93] (S3: theme, gg)	List of length 93
coordinates	environment [5] (S3: CoordCartesian, Cc)	<environment: 0x000002477d58bbf8>
facet	environment [2] (S3: FacetNull, Facet, gg)	<environment: 0x000002477d58a538>
plot_env	environment [25]	<environment: R_GlobalEnv>
labels	list [4]	List of length 4

Figure 13: The List generated by ggplot() contains environment objects

## 7.5 Changing values in the plot List

We can both extract values from a List and change the values. In this case, we can make changes to the plot by changing values within the List.

Let's first make a copy of the plot and makes changes on the copy:

```
plot2 = plot1;
```

The changes we will make are:

1. change the text color to red
2. change the x-axis label to use the degree symbol instead of "degree"
  - `\u00B0` means get the [Unicode character](#) designated `00B0` (which is a degree symbol)
3. change the axis lines to dashed lines
4. increase the size of the axis lines

```
plot2$theme$text$colour = "red";
plot2$labels$x = "Average Temperature (\u00B0 F)";
plot2$theme$line$linetype = 2;
plot2$theme$line$size = 0.8;
```

And plot the results

```
plot(plot2);
```

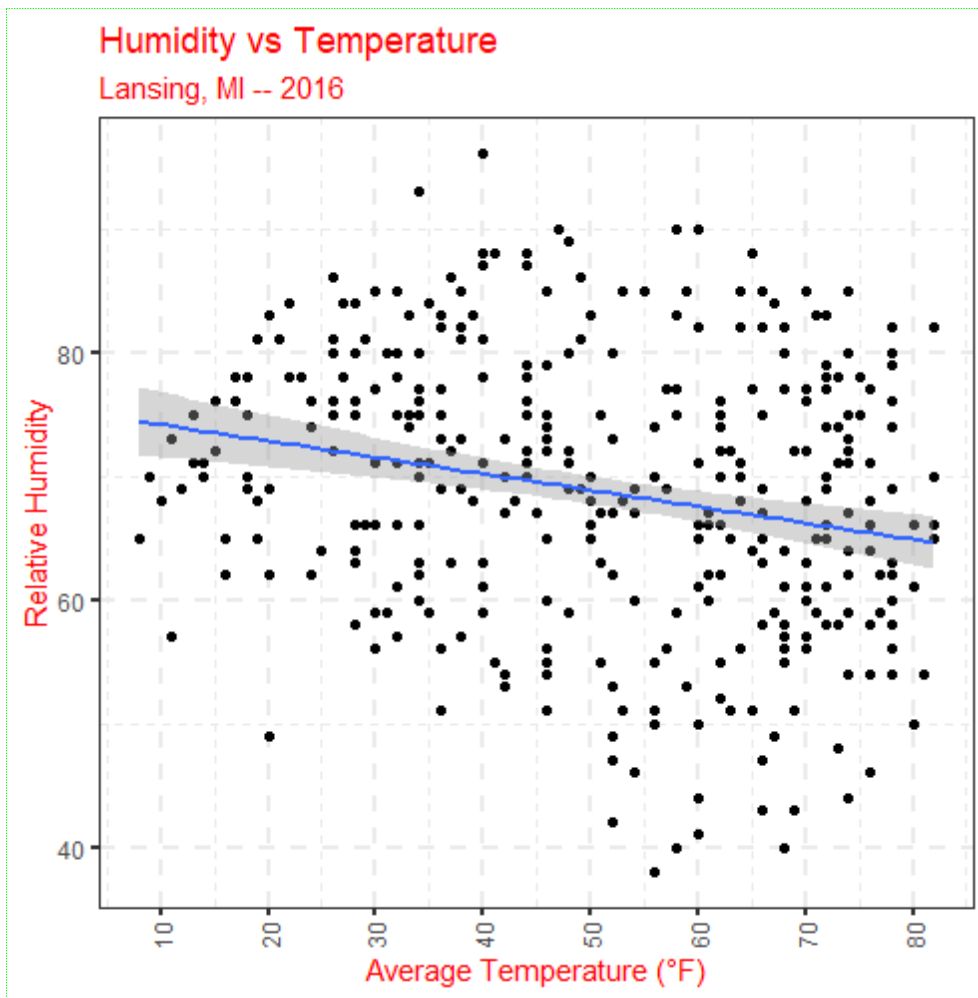


Figure 14: The plot after changes were made to the List

Of course, all of these changes can be made within the ***ggplot()*** function call, which is most likely what you would want to do. But, the List object contains everything that went into making the plot so it can be modified to tweak the plot.

---

## 8 Application

---

A) Why is it better to subset objects by name instead of position numbers?

- Save the answer to a 1-character vector
- Add the ***name*** "comment" to the value

B) Using ***weatherData***, create a linear model of dewpoint vs temperature

- Save the model to a List named ***linearModel1***

C) Using only values in ***linearModel1***, find and save:

- the slope and the intercept
- the last 10 effects values
- the first 10 values from the original temperature vector



- the most extreme residual value (i.e., furthest from 0 – could be positive or negative)

D) Using the data from ***Lansing2016Noaa.csv***, create a scatterplot of dewpoint vs average temperature

- include a linear regression of dewpoint vs. average temperature
- save the plot information to a List named ***scatterPlot1***

E) Make a copy of the List variable ***scatterPlot1*** named ***scatterPlot2***

- make 1 text change and 2 style changes by changing values in the List variable ***scatterPlot2***
- do not make the same changes that were made in the lesson

F) Subset an object in the GGPlot list that is exactly 6 levels deep.

- avoid subsetting any Name that are italicized
- save the object to a variable named ***ggplotObj***

G) Create one List named ***app2\_12\_List*** that contains the objects from parts A-F in this application

Save the script as ***app2-12.r*** in your ***scripts*** folder and email your Project Folder to Charlie Belinsky at belinsky@msu.edu.

[Instructions for zipping the Project Folder are here.](#)

If you have any questions regarding this application, feel free to email them to Charlie Belinsky at belinsky@msu.edu.

---

## 8.1 Questions to answer

Answer the following in comments inside your application script:

1. What was your level of comfort with the lesson/application?
2. What areas of the lesson/application confused or still confuses you?
3. What are some things you would like to know more about that is related to, but not covered in, this lesson?