

Lecture 1.8 – Lists and Data Frames

Specific Learning Objectives:

OMG it's a study guide!!!!

1.1.10 – Create vectors, arrays, matrices, lists, and data frames.

1.1.11 – Understand vectors and vectorized calculations.

1.1.12 – Learn how to index vectors, arrays, matrices, lists, and data frames.

Data sets in base R



- Base R comes with several sample data sets.
- To view: `> data()`

```
R data sets x
Data sets in package 'datasets':

AirPassengers      Monthly Airline Passenger Numbers 1949-1960
BJsales            Sales Data with Leading Indicator
BJsales.lead (BJsales) Sales Data with Leading Indicator
BOD                Biochemical Oxygen Demand
CO2                Carbon Dioxide Uptake in Grass Plants
ChickWeight        Weight versus age of chicks on different diets
DNase              Elisa assay of DNase
EuStockMarkets     Daily Closing Prices of Major European Stock Indices,
                  1991-1998
Formaldehyde       Determination of Formaldehyde
HairEyeColor       Hair and Eye Color of Statistics Students
Harman23.cor       Harman Example 2.3
Harman74.cor       Harman Example 7.4
Indometh           Pharmacokinetics of Indomethacin
InsectSprays       Effectiveness of Insect Sprays
JohnsonJohnson    Quarterly Earnings per Johnson & Johnson Share
LakeHuron          Level of Lake Huron 1875-1972
LifeCycleSavings   Intercountry Life-Cycle Savings Data
Loblolly           Growth of Loblolly pine trees
Nile               Flow of the River Nile
Orange             Growth of Orange Trees
```

Data sets in base R



- To load into RStudio's environment:

```
> data(dataset.name)
> force(dataset.name)
```

```
> data(BOD)
> force(BOD)
```

	Time	demand
1	1	8.3
2	2	10.3
3	3	19.0
4	4	16.0
5	5	15.6
6	7	19.8

```
> |
```

```
> data("ToothGrowth")
> force(ToothGrowth)
```

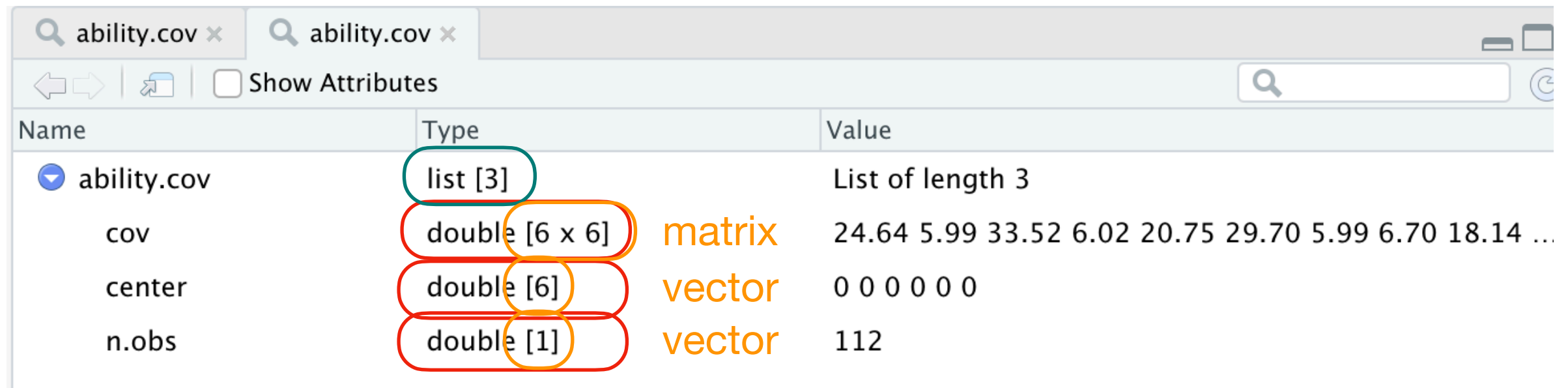
	len	supp	dose
1	4.2	VC	0.5
2	11.5	VC	0.5
3	7.3	VC	0.5
4	5.8	VC	0.5
5	6.4	VC	0.5
6	10.0	VC	0.5
7	11.2	VC	0.5

Lists and Data Frames — What's the difference?

- Two very common classes that deal with *mixed* data types are **lists** and **data frames**.
 - Mixed data types would include different types of data in the same object (ex: characters, integers, and numerics).
 - Both lists and data frames can handle mixed data types, but they come with some differences.
 - To compare and contrast, load the following data sets into your R Studio environment:
 - ▶ List example:
`ability.cov`
 - ▶ Data frame example:
`ToothGrowth`

Characteristics of Lists

- List example shows a few characteristics of lists.

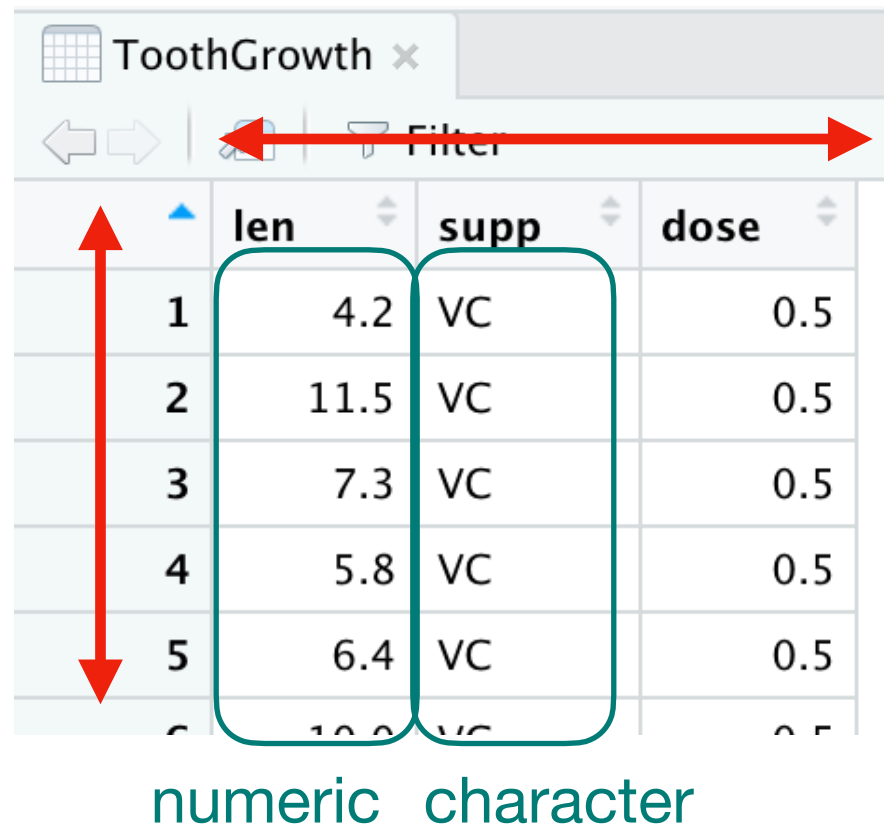


Name	Type	Value
ability.cov	list [3]	List of length 3
cov	double [6 x 6] matrix	24.64 5.99 33.52 6.02 20.75 29.70 5.99 6.70 18.14 ...
center	double [6] vector	0 0 0 0 0 0
n.obs	double [1] vector	112

- Lists can have as many members as you want (this example has 3)
- Each member in a list can be a different size
- Each member can be a different class (but within an element, they must be the same type/class).
- Each member can be different data type!
- (not shown) A member can even be another list!

Characteristics of Data Frames

- Data frame example shows a few characteristics of data frames.



	len	supp	dose
1	4.2	VC	0.5
2	11.5	VC	0.5
3	7.3	VC	0.5
4	5.8	VC	0.5
5	6.4	VC	0.5







numeric character

- Data frames are limited to 2D structures (like matrices).
- Data frames can have different data types in different columns (but must have the same type within a column)
- Data frames can have as many columns and rows as you'd like, but all columns must have the same number of rows and all rows have to have the same number of columns.

Check Your Understanding

Which class of object would you use if you needed:

- a) Members of different sizes
- b) Members of different classes
- c) Both a and b

List	Data frame
	
	
	

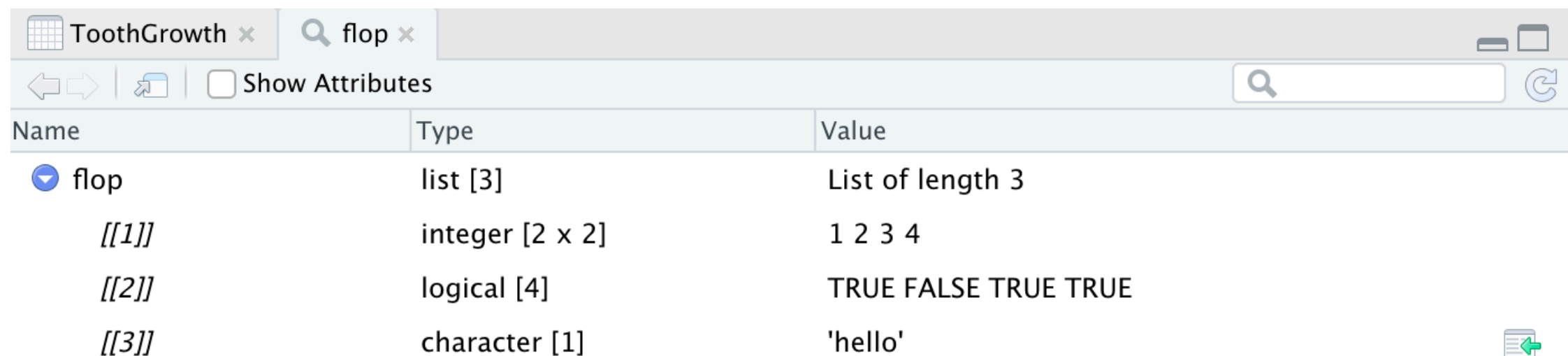
Creating and Viewing Lists

- Creating lists is easy with `list()`

```
> flop <- list(matrix(data=1:4, nrow=2, ncol=2), c(T,F,T,T), "hello")
```

- Inspect your list using `View()`

```
> View(flop)
```



Name	Type	Value
▼ flop	list [3]	List of length 3
[[1]]	integer [2 x 2]	1 2 3 4
[[2]]	logical [4]	TRUE FALSE TRUE TRUE
[[3]]	character [1]	'hello'

- Inspect individual members of your list using the button
- The button will give you the code for reference of member.

Member Reference of Lists

- There are several ways of calling individual elements of lists.

ToothGrowth x flop x		
← → Show Attributes		
Name	Type	Value
▼ flop	list [3]	List of length 3
[[1]]	integer [2 x 2]	1 2 3 4
[[2]]	logical [4]	TRUE FALSE TRUE TRUE
[[3]]	character [1]	'hello'

- Double brackets give access to members of the list:

```
> flop[[3]]  
[1] "hello"
```

```
> flop[[1]]  
      [,1] [,2]  
[1,]    1    3  
[2,]    2    4
```

- Adding single brackets will reference individual elements of the member:

```
> flop[[2]][2]  
[1] FALSE
```

```
> flop[[1]][4]  
[1] 4
```

Member Reference of Lists

- If you want multiple members from a list, you can perform list slicing by using single brackets with a vector:

```
> flop[c(2,3)]  
[[1]]  
[1] TRUE FALSE TRUE TRUE  
  
[[2]]  
[1] "hello"
```

**Pulls both members 2
and 3**

- For easy reference, you can also name members of your list:

```
> names(flop) <- c("mymatrix", "mylogicals", "mystring")
```

Name	Type	Value
flop	list [3]	List of length 3
mymatrix	integer [2 x 2]	1 2 3 4
mylogicals	logical [4]	TRUE FALSE TRUE TRUE
mystring	character [1]	'hello'

- Reference using a dollar sign:

```
> flop$mylogicals  
[1] TRUE FALSE TRUE TRUE
```

```
> flop$mymatrix[1,1]  
[1] 1
```

List Info

- Information for lists is a bit different than for other classes

- If you use length, you will get the number of members:

```
> length(flop)
[1] 3
```

- Most other functions will return NULL if you use them on lists:

```
> dim(flop)
NULL
```

- But you can use the same functions on individual members for info:

```
> dim(flop$mymatrix)
[1] 2 2
```

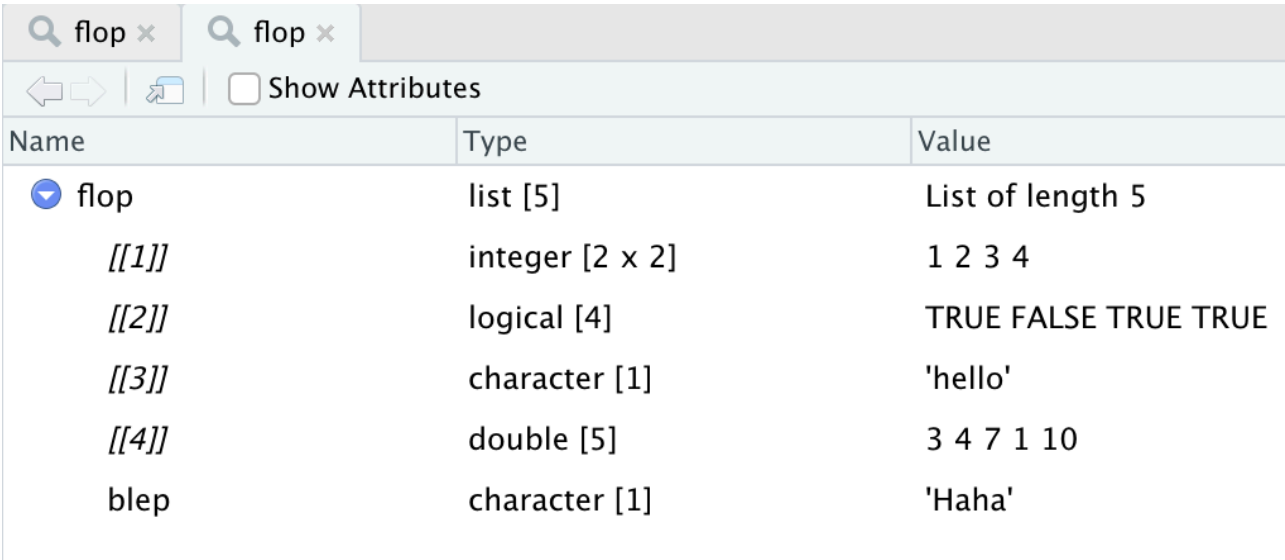
```
> length(flop$mystring)
[1] 1
```

Modifying Lists

- Modifying lists is easy!
- Add members by simply assigning a new number or named member to the list:

```
> flop[[4]]<- c(3,4,7,1,10)
```

```
> flop$blep <- "Haha"
```



The screenshot shows the R Studio environment with two tabs labeled 'flop'. The 'Environment' pane displays the 'flop' object as a list of length 5. The 'Value' column shows the contents of the list: a 2x2 integer matrix, a 4-element logical vector, a single character string, a 5-element double vector, and a single character string.

Name	Type	Value
flop	list [5]	List of length 5
[[1]]	integer [2 x 2]	1 2 3 4
[[2]]	logical [4]	TRUE FALSE TRUE TRUE
[[3]]	character [1]	'hello'
[[4]]	double [5]	3 4 7 1 10
blep	character [1]	'Haha'

- Adding elements to each member depends on the data type of the member!

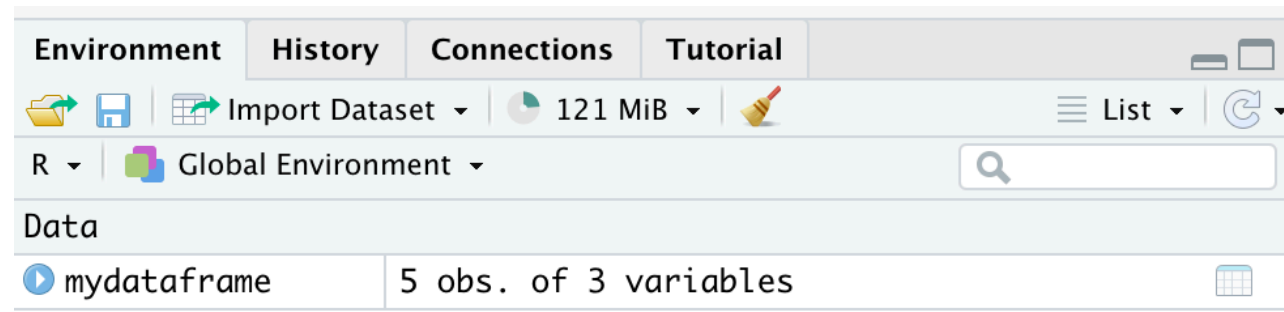
Check Your Understanding

Create a list in which each member contains one of each data types you've learned so far in the course!

Creating Data Frames

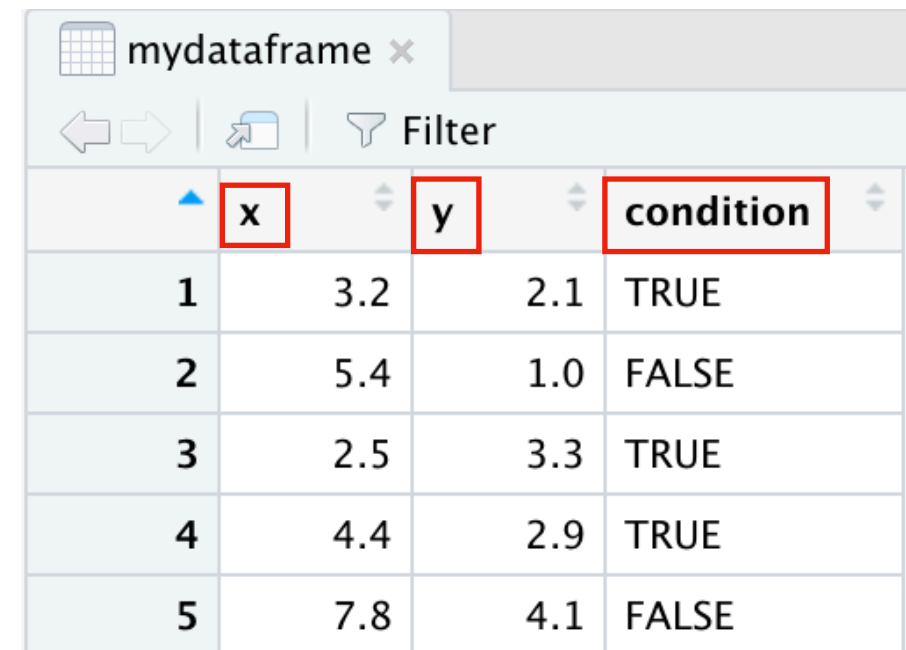
- Create a data frame using `data.frame()`

```
> mydataframe <- data.frame(x=c(3.2, 5.4, 2.5, 4.4, 7.8), y=c(2.1, 1.0, 3.3, 2.9, 4.1),  
+ condition=c(T,F,T,T,F))
```



The R Studio Environment pane shows the 'mydataframe' object in the Global Environment. It indicates that the object contains 5 observations of 3 variables. The pane also shows tabs for Environment, History, Connections, and Tutorial, along with a search bar and a 'List' button.

Environment	History	Connections	Tutorial
R	Global Environment	121 MiB	
Data			
mydataframe		5 obs. of 3 variables	



A visual representation of the 'mydataframe' data frame, showing a table with 5 rows and 4 columns. The columns are labeled 'x', 'y', and 'condition'. The rows are numbered 1 to 5. The values for 'x' are 3.2, 5.4, 2.5, 4.4, and 7.8. The values for 'y' are 2.1, 1.0, 3.3, 2.9, and 4.1. The values for 'condition' are TRUE, FALSE, TRUE, TRUE, and FALSE.

	x	y	condition
1	3.2	2.1	TRUE
2	5.4	1.0	FALSE
3	2.5	3.3	TRUE
4	4.4	2.9	TRUE
5	7.8	4.1	FALSE

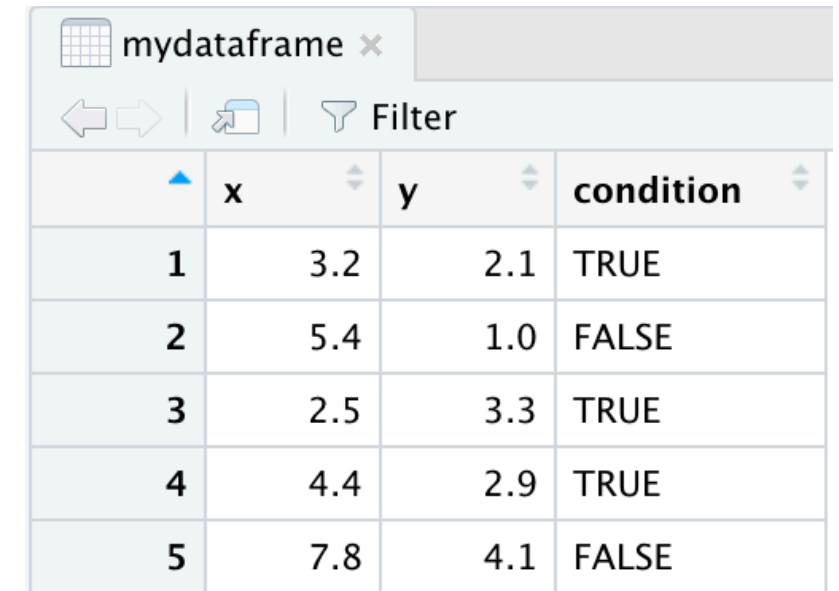
Data frames report observations (or rows) and variables (or columns)

- You can set column names during data-frame construction!
- Rename column names using `colnames()`

```
> colnames(mydataframe) <- c("up", "down", "cow")
```

Indexing Data Frames

- There are a LOT of ways of indexing data within data frames.
- Similar to matrices, you can specify row and column positions with square brackets:



	x	y	condition
1	3.2	2.1	TRUE
2	5.4	1.0	FALSE
3	2.5	3.3	TRUE
4	4.4	2.9	TRUE
5	7.8	4.1	FALSE

```
> mydataframe[2,3]  
[1] FALSE
```

```
> mydataframe[2,2]  
[1] 1
```

- You can also use double square brackets to call columns.

```
> mydataframe[["y"]]  
[1] 2.1 1.0 3.3 2.9 4.1
```

```
> mydataframe[["y"]][4]  
[1] 2.9
```

- You can *also* use a dollar sign to call columns.

```
> mydataframe$y  
[1] 2.1 1.0 3.3 2.9 4.1
```

```
> mydataframe$y[4]  
[1] 2.9
```

Data-frame Indexing Alignment Chart

```
df <- data.frame(x = 1:3, y = 4:6)
```

LAWFUL GOOD

```
df[["y"]]
```

NEUTRAL GOOD

```
pull(df, y)
```

CHAOTIC GOOD

```
select(df, y)[[1]]
```

LAWFUL NEUTRAL

```
df[, "y"]
```

TRUE NEUTRAL

```
df$y
```

CHAOTIC NEUTRAL

```
df[, names(df)=="y"]
```

LAWFUL EVIL

```
select_at(df,  
vars(matches("^y$")))[[1]]
```

NEUTRAL EVIL

```
df[, 2]
```

CHAOTIC EVIL

```
pmap_int(df, function(x,y){y})
```

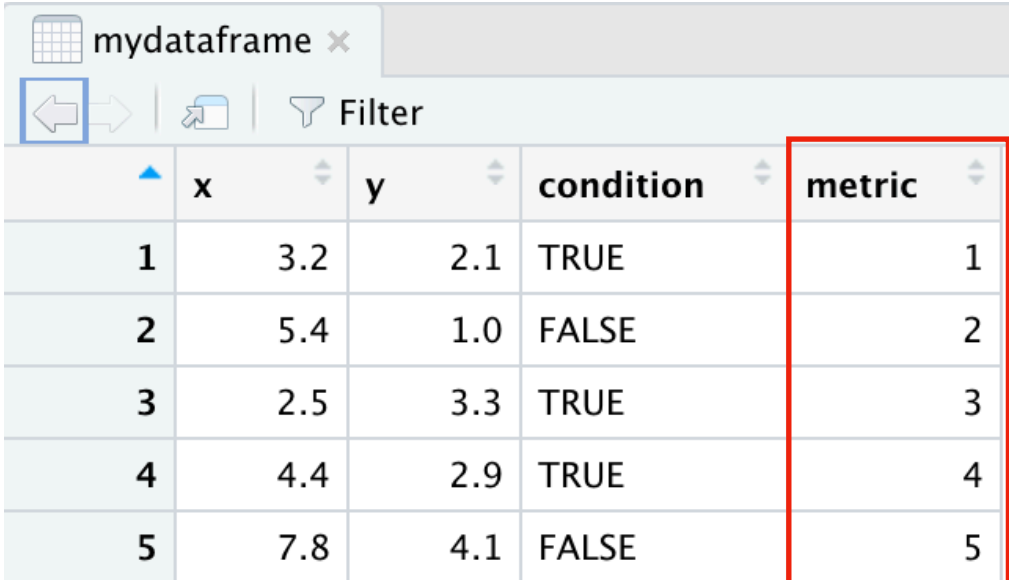

Modifying Data Frames

- Modifying data frames is similar to list, but also has some additional helpful functions.

- Add a column by assigning it with a new name:

```
> mydataframe$metric <- seq(1,5)
```

(Remember, the new column has to be the same length as the number of rows in the data frame!)



	x	y	condition	metric
1	3.2	2.1	TRUE	1
2	5.4	1.0	FALSE	2
3	2.5	3.3	TRUE	3
4	4.4	2.9	TRUE	4
5	7.8	4.1	FALSE	5

- Add multiple rows by using “row bind” **rbind()**:

```
> mydataframe <- rbind(mydataframe, mydataframe)
```

Binds two copies of mydataframe together!

- Add multiple columns by using “column bind” **cbind()**:

```
> mydataframe <- cbind(mydataframe, mydataframe)
```

Info on Data Frames

- There are some helpful functions for data frames that will make life a bit easier!
- Print out the first few lines of a data frame: **head()**
- Print out the last few lines of a data frame: **tail()**

```
> head(ToothGrowth)
```

	len	supp	dose
1	4.2	VC	0.5
2	11.5	VC	0.5
3	7.3	VC	0.5
4	5.8	VC	0.5
5	6.4	VC	0.5
6	10.0	VC	0.5

**(Note: these also
work with other
data classes)**

```
> tail(ToothGrowth)
```

	len	supp	dose
55	24.8	OJ	2
56	30.9	OJ	2
57	26.4	OJ	2
58	27.3	OJ	2
59	29.4	OJ	2
60	23.0	OJ	2

- Print out a summary of elements in the data frame:
summary()

```
> summary(ToothGrowth)
```

	len	supp		dose
Min.	: 4.20	OJ:30	Min.	:0.500
1st Qu.:	13.07	VC:30	1st Qu.:	0.500
Median	:19.25		Median	:1.000
Mean	:18.81		Mean	:1.167
3rd Qu.:	25.27		3rd Qu.:	2.000
Max.	:33.90		Max.	:2.000

**Automatically
summarizes each
column!**

**Summary statistics
appropriate for data
type!**

Check Your Understanding

In the `ToothGrowth` data set, how can you print out all the measured tooth lengths from their study?

How can you find the mean and standard deviation of these lengths?

Logical Record Subsets

- Information can also be pulled based on a logical test for most data structures (lists and data frames included!)
 - This is really useful if you don't care what the element position is, you care more about the value of the elements.

You run the following line to create a matrix:

```
my.matrix2 <- matrix(seq(1,21), nrow=7, byrow=TRUE)
```

Which line of code will subset only two-digit numbers (those greater than 9)?

- one option: `my.matrix2[4:7,]`
- another option: `my.matrix2[my.matrix2>9]`

```
> my.matrix2>9
      [,1] [,2] [,3]
[1,] FALSE FALSE FALSE
[2,] FALSE FALSE FALSE
[3,] FALSE FALSE FALSE
[4,]  TRUE  TRUE  TRUE
[5,]  TRUE  TRUE  TRUE
[6,]  TRUE  TRUE  TRUE
[7,]  TRUE  TRUE  TRUE
```

```
> my.matrix2[my.matrix2>9]
[1] 10 13 16 19 11 14 17 20 12 15 18 21
```

Logical Record Subsets

- In data frames, be a little careful to specify whether you are looking for the record in a column or a row.
 - The easiest way to do this is to come up with the test first and then where it is searching second.

Example: In the **ToothGrowth** data set, find all the data associated with animals given orange juice as their supplement.

- We want to first create the test. How can we restrict data to only orange juice (OJ)?

```
ToothGrowth$supp == "OJ"
```

- Now use this logical vector to retrieve the records:

```
ToothGrowth[ToothGrowth$supp == "OJ", ]
```

**Get all the rows
where this statement
is true**

**And get ALL
columns for those
rows!**

Check Your Understanding

In the `ToothGrowth` data set, how can you print out all the measured tooth lengths from their study that were only given a dose of 1.0?

How can you find the mean and standard deviation of these lengths?

Can you find the mean of the tooth lengths for animals given an OJ dose of 1.0?

Action Items

- 1. Complete Assignments 1.10 and 1.11.**
- 2. Read Davies Ch. 6 for next time.**