STA130 - Class #3: How R You?

Nathan Taback

2018-01-22

Today's Class

- RStudio user interface
- R Objects
- R Functions
- R Scripts
- R Packages
- R Lists
- R Notation
- R Missing Data
- dplyr

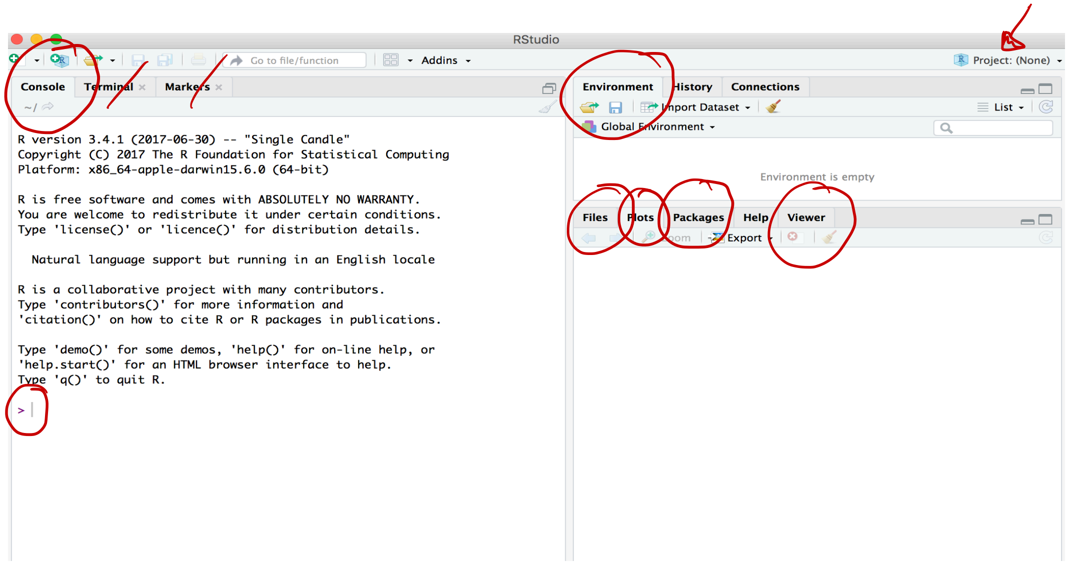
Announcements

- Tutorial grades will be assigned according to the following marking scheme.

	Mark
Attendance for the entire tutorial	1
Assigned homework completion[a]	1
In-class exercises	4
Total	6

- You will learn about the mentorship program in this week's tutorial (3% of final grade).

RStudio User Interface



```
R version 3.4.1 (2017-06-30) -- "Single Candle"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

R Objects

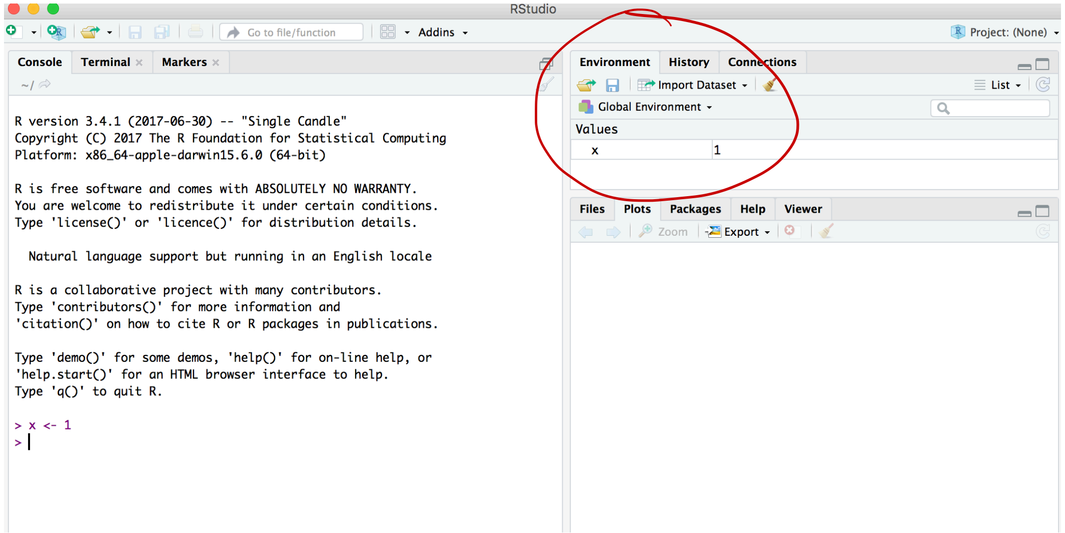
- R lets you save data by storing it inside an R object.
- What's an object? Just a name that you can use to call up stored data.

```
x <- 1
x
```

```
## [1] 1
```

Environment Pane in RStudio

- When you create an object, the object will appear in the environment pane of RStudio.



```
R version 3.4.1 (2017-06-30) -- "Single Candle"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> x <- 1
>
```

Values

x	1

Functions

$$abs(x) = |x| = \begin{cases} x, & x \geq 0 \\ -x, & x < 0 \end{cases}$$

- R comes with many functions that you can use to do sophisticated tasks like random sampling.
- For example, you can round a number with the round function `round()`, or calculate its absolute value with `abs()`.
- Write the name of the function and then the data you want the function to operate on in parentheses:

```
round(-2.718282, 2)
```

```
## [1] -2.72
```

```
abs(-5)
```

```
## [1] 5
```

$f \quad g$

```
abs(round(-2.718282, 2))
```

$$\rightarrow abs(-2.72) = 2.72.$$

```
## [1] 2.72
```

$$(f \circ g)(x) = f(g(x))$$

Function Constructor

- Every function in R has three basic parts: a name, a body of code, and a set of arguments.
- To make your own function, you need to replicate these parts and store them in an R object, which you can do with the function function.
- To do this, call `function()` and follow it with a pair of braces, `{}`: `my_function <- function() {}`

```
my_function <- function ( ) {
        x <- 1
        x
      }
```

my_function ( ) — assumes no arguments.

Function Constructor

Set. Seed (1)

- We can simulate rolling a pair of dice and adding the result with the code:

```
die <- 1:6
dice <- sample(die, size = 2, replace = TRUE)
sum(dice)
```

```
## [1] 9
```

1,2,3,4,5,6 die ← c(1,2,3,4,5,6)

Chooses two numbers from 1,2,3,4,5,6 Such that:

If the first value is, Say, 1, then the Second value is selected from 1,2,3,4,5,6

1,1 6,6

2 – 12

is the range of values for Sum(dice).

Function Constructor

- We can create our own function with

```
my_func <- function() {
     code.
     }
```

```
roll <- function() {
  die <- 1:6
  dice <- sample(die, size = 2, replace = TRUE)
  sum(dice)
}
```

Call the function `roll()`

```
roll() # call the function.  NB: result will differ with every call
```

```
## [1] 4
```

Function Arguments

Instead of rolling one die consider rolling four or ten dice then adding the results of all the rolls together.

```
                                    argument of the function
roll2 <- function(numrolls) { # ~~x~~ is the argument of the function roll2
                                 numrolls
  die <- 1:6
  dice <- sample(die, size = numrolls, replace = TRUE) # the size of the sample
  sum(dice) # add up the roll results
}
```

`numrolls` is called an *argument* of the function `roll2()`.

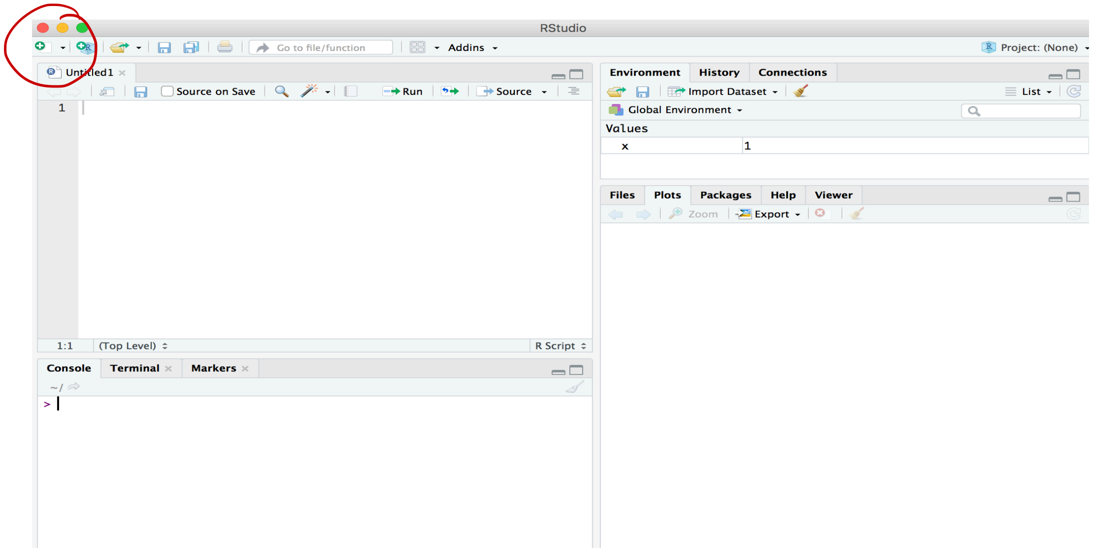
Let's simulate rolling ten dice and adding the results together.

```
roll2(10)
```

```
## [1] 40
```

Scripts

- If we want to edit the function `roll2()` then we will want to save it in a script.
- To do this in RStudio File > New File > R script in the menu bar.

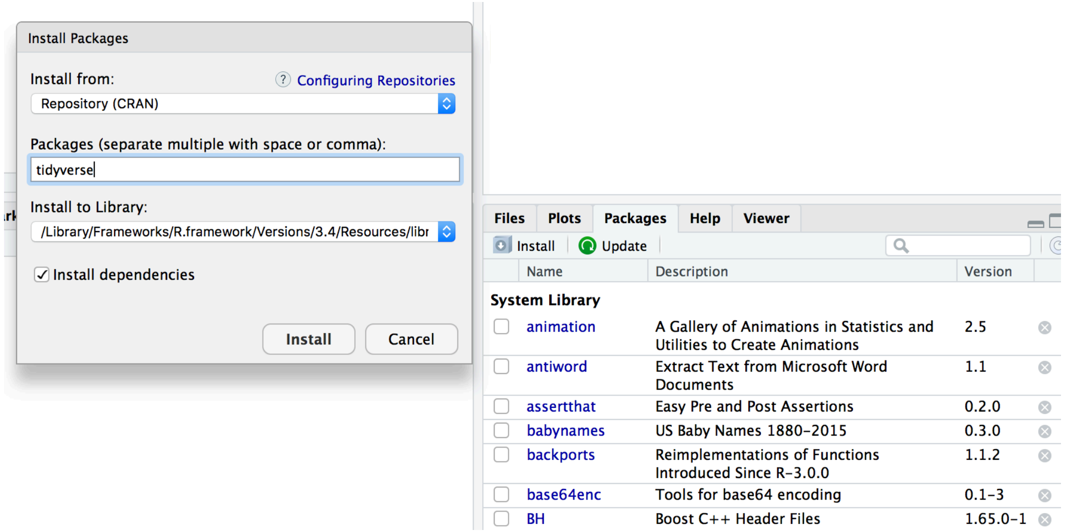


Packages

- You’re not the only person writing your own functions with R.
- Many professors, programmers, and statisticians use R to design tools that can help people analyze data.
- They then make these tools free for anyone to use.
- To use these tools, you just have to download them. They come as preassembled collections of functions and objects called packages.
- We have already used two packages `ggplot2` and `dplyr`.

Packages

To install the package `tidyverse` in RStudio go to the Packages tab in RStudio and click Install.



Install Packages

Install from: Repository (CRAN)

Packages (separate multiple with space or comma): tidyverse

Install to Library: /Library/Frameworks/R.framework/Versions/3.4/Resources/libr

☑ Install dependencies

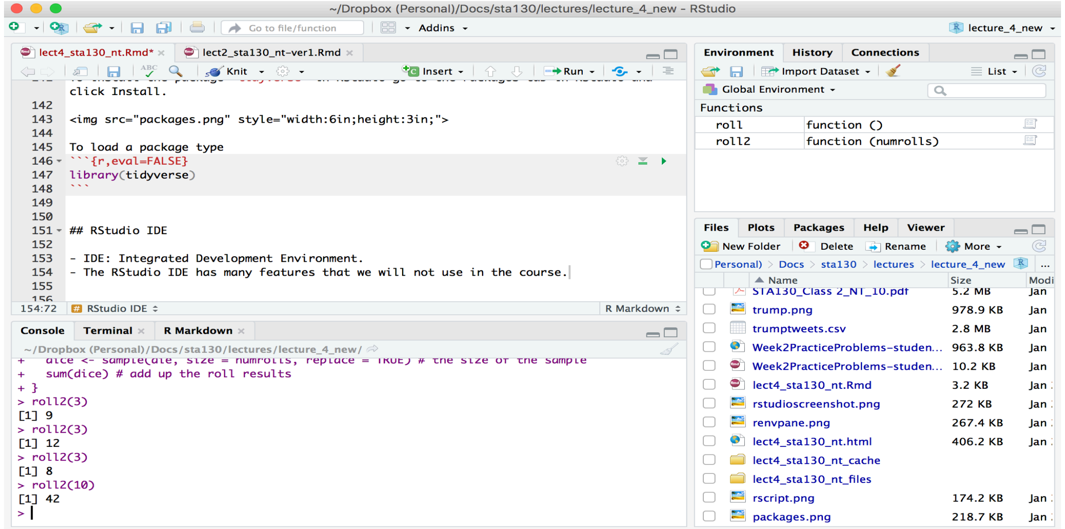
	Name	Description	Version
	System Library		
□	animation	A Gallery of Animations in Statistics and Utilities to Create Animations	2.5
□	antiword	Extract Text from Microsoft Word Documents	1.1
□	assertthat	Easy Pre and Post Assertions	0.2.0
□	babynames	US Baby Names 1880-2015	0.3.0
□	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.2
□	base64enc	Tools for base64 encoding	0.1-3
□	BH	Boost C++ Header Files	1.65.0-1

To load a package type

```
library(tidyverse)
```

RStudio IDE

- IDE: Integrated Development Environment.
- The RStudio IDE has many features that we will not use in the course.



~~~
To install the package `tidyverse` in RStudio go to the Packages tab in RStudio and
click Install.

<img src="packages.png" style="width:6in;height:3in;">

To load a package type
```{r,eval=FALSE}
library(tidyverse)
```


## RStudio IDE

- IDE: Integrated Development Environment.
- The RStudio IDE has many features that we will not use in the course.
~~~

~/Dropbox (Personal)/Docs/sta130/lectures/lecture_4_new/

```
+   dice <- sample(die, size = numrolls, replace = TRUE) # the size of the sample
+   sum(dice) # add up the roll results
+ }
> roll2(3)
[1] 9
> roll2(3)
[1] 12
> roll2(3)
[1] 8
> roll2(10)
[1] 42
> 
```

Functions

roll	function ()
roll2	function (numrolls)

Personal) > Docs > sta130 > lectures > lecture_4_new

Name	Size	Modi
STA130_Class 2_NT_10.pdf	5.2 MB	Jan
trump.png	978.9 KB	Jan
trumptweets.csv	2.8 MB	Jan
Week2PracticeProblems-studen...	963.8 KB	Jan
Week2PracticeProblems-studen...	10.2 KB	Jan
lect4_sta130_nt.Rmd	3.2 KB	Jan
rstudioscreenshot.png	272 KB	Jan
renvpane.png	267.4 KB	Jan
lect4_sta130_nt.html	406.2 KB	Jan
lect4_sta130_nt_cache		
lect4_sta130_nt_files		
rscript.png	174.2 KB	Jan
packages.png	218.7 KB	Jan

- The **console** is where you can type an R command at the prompt and the result is returned.
- Write code in an R script, R Markdown document, or R Notebook.
- Run a script or R chunks from an R Markdown or R Notebook by pushing the run button in the chunk.

R Objects

- R stores data in objects such as vectors, arrays, and matricies.
- In most applications we will ususally load data from an external file.

R Objects - Atomic Vectors

You can make an atomic vector by grouping some values of data together with c:

```
die<-c(1,2,3,4,5,6)
die
```

```
## [1] 1 2 3 4 5 6
```

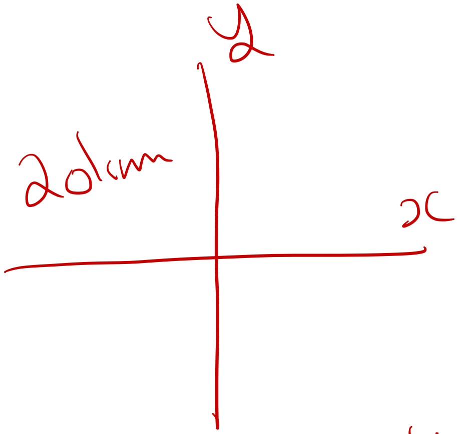
```
is.vector(die)
```

```
## [1] TRUE
```

```
length(die)
```

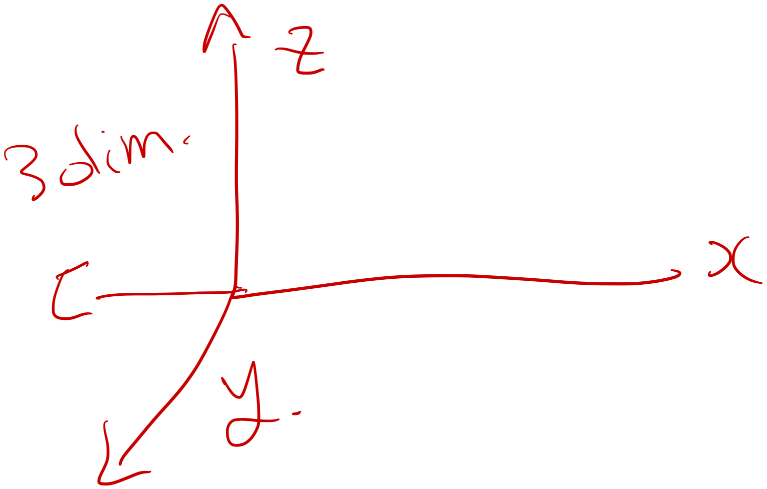
```
## [1] 6
```

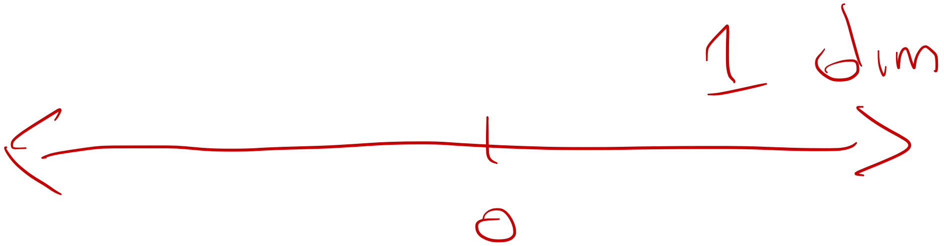
built in functions in R.



△ ← (x, y)

(x, y, z)





R Objects - Atomic Vectors

You can also make an atomic vector with just one value. R saves single values as an atomic vector of length 1:

```
two <- 2
two
```

```
## [1] 2
```

R Objects - Atomic Vectors: Integer and Character

- Each atomic vector can only store one type of data. You can save different types of data in R by using different types of atomic vectors.

- R recognizes six basic types of atomic vectors: (doubles, integers, characters, logicals) complex, and raw.

  $i = \sqrt{-1}$

- We will not be using complex or raw types in STA130.

- Integer vectors included a capital L with input, and character vectors have input surounded by quotation marks.

R Objects - Atomic Vectors: Integer and Character

```
mynums <- c(2L,3L)
courses <- "STA130"
courses <- c("STA130", "MAT137")
sum(mynums)
```

```
## [1] 5
```

```
sum(courses)
```

```
## Error in sum(courses): invalid 'type' (character) of argument
```

```
sum(courses == "STA130")
```

① Coercion.

∵ Courses == "STA130"

is True

and TRUE ~~is~~ always 1

```
## [1] 1
```

R Objects - Double Vectors

- A double vector stores real numbers. Doubles are often called numerics.

```
die <- c(1,2,3,4,5,6)
typeof(die)
```

```
## [1] "double"
```

R Objects - Logical Vectors

- Logical vectors store TRUEs and FALSEs, R's form of Boolean data. Logicals are very helpful for doing things like comparisons:

```
3 > 4
```

```
## [1] FALSE
```

- TRUE or FALSE in capital letters (without quotation marks) will be treated as logical data. R also assumes that T and F are shorthand for TRUE and FALSE.

```
logic <- c(TRUE, FALSE, TRUE)
logic
```

```
## [1]  TRUE FALSE  TRUE
```

R Objects - Atomic Vectors: `dim()`

You can transform an atomic vector into an n-dimensional array by giving it a dimen- sions attribute with dim.

# of rows

# of Cols.

```
die <- c(1,2,3,4,5,6)
dim(die) <- c(2,3) # a 2x3 matrix
die
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$ 2x2 matrix

Notation allows you to access Values in the matrix

a_{ij} - Value in the i^{th} row, j^{th} Column.

die[, 3]

blank

return Values of 5, 6.

```
die <- c(1,2,3,4,5,6)
dim(die) <- c(3,2) # a 3x2 matrix
die
```

rows

cols.

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

die[1,]

return 1,3,5

R always fills up each matrix by columns, instead of by rows unless you use `matrix()` or `array()`.

Factors

- Factors are R's way of storing categorical information, like ethnicity or eye color.
- A factor as something like sex since it can only have certain values.
- Factors very useful for recording the treatment levels of a categorical variable.

```
sex <- factor(c("male", "female", "female", "male"))
typeof(sex)
```

```
## [1] "integer"
```

```
unclass(sex) # shows how R is storing the factor vector
```

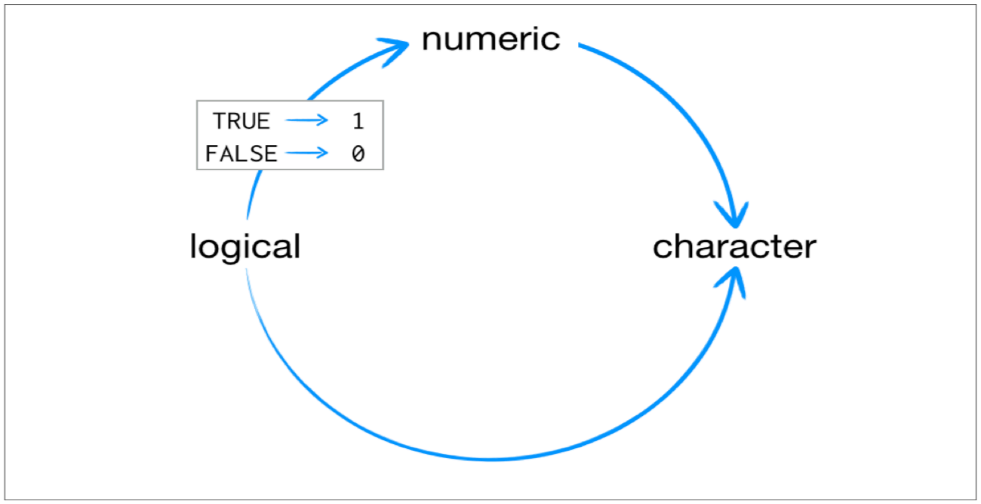
male female

```
## [1] 2 1 1 2
## attr(,"levels")
## [1] "female" "male"
```

$$\text{Sex} = \begin{cases} 2 & \text{if "female"} \\ 1 & \text{if "male"} \end{cases}$$

Coercion

R always follows the same rules when it coerces data types. Once you are familiar with these rules, you can use R's coercion behavior to do surprisingly useful things.



1 -> "1"

For example `sum(c(TRUE, TRUE, FALSE, FALSE))` will become `sum(c(1, 1, 0, 0))`.

```
sum(c(TRUE, TRUE, FALSE, FALSE))
```

```
## [1] 2
```

Lists

- Lists are like atomic vectors because they group data into a one-dimensional set.
- Lists do not group together individual values.
- Lists group together R objects, such as atomic vectors and other lists.
- For example, you can make a list that contains a numeric vector of length 31 in its first element, a character vector of length 1 in its second element, and a new list of length 2 in its third element.

1dim character vector

```
list1 <- list(1:31, "Prof. Taback", list(TRUE, FALSE))
list1
```

31 dim vector

2dim logical vector.

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31
##
## [[2]]
## [1] "Prof. Taback"
##
## [[3]]
## [[3]][[1]]
## [1] TRUE
##
## [[3]][[2]]
## [1] FALSE
```

list1[[1]] First elementof of list.

list 1[[2]] Second elementof list.

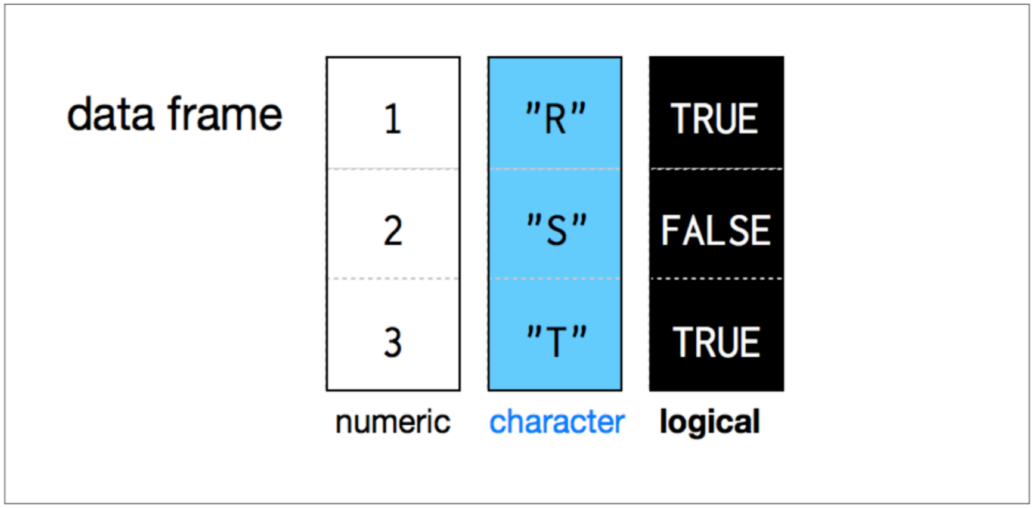
list1[[3]][[3]][[1]]
return TRUE

Data Frames

- Data frames are the two-dimensional version of a list.
- They are the most useful storage structure for data analysis
- A data frame is R’s equivalent to the Excel spreadsheet because it stores data in a similar format.

Data Frames

- Data frames group vectors together into a two-dimensional table.
- Each vector becomes a column in the table.
- As a result, each column of a data frame can contain a different type of data; but within a column, every cell must be the same type of data.



data frame

1	"R"	TRUE
2	"S"	FALSE
3	"T"	TRUE
numeric	character	logical

Data Frames

```
student_num <- c(1, 2, 3, 4)
name <- c("Nadia", "Shiyi", "Yizhe", "Wei")
mydat <- data.frame(obsnum = student_num, student_name = name)
mydat
```

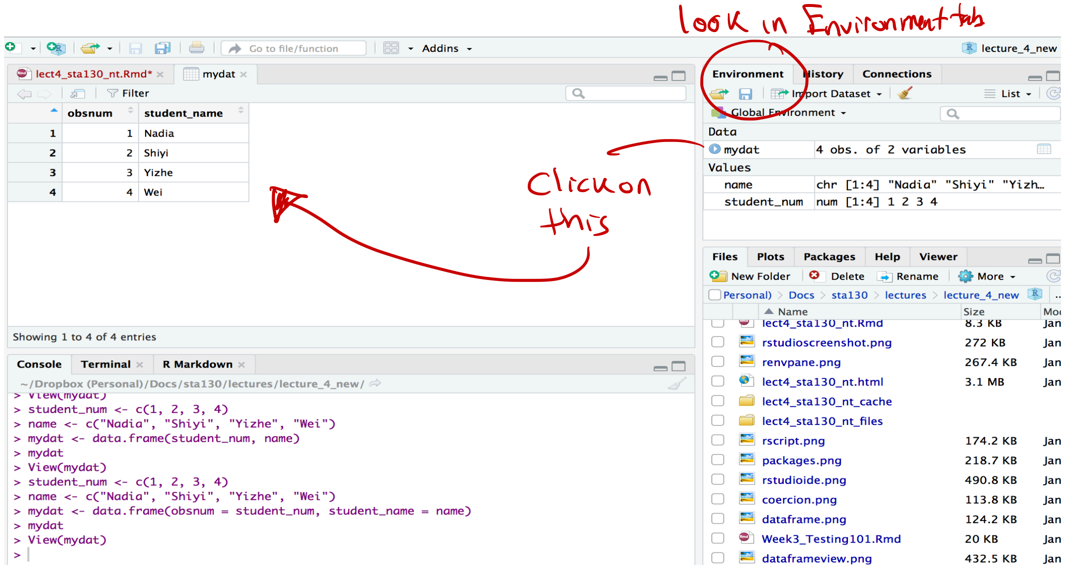
↳ rename first variable as obsnum

```
##   obsnum student_name
## 1      1        Nadia
## 2      2        Shiyi
## 3      3        Yizhe
## 4      4          Wei
```

- Creating a data frame by hand takes a lot of typing, but you can do it with the `data.frame()` function.
- Give `data.frame()` any number of vectors, each separated with a comma.
- Each vector should be set equal to a name that describes the vector.
- `data.frame()` will turn each vector into a column of the new data frame.

Data Frames

You can view a data frame in RStudio by clicking on the data frame name in the Environment tab



look in Environment tab

Click on this

	obsnum	student_name
1	1	Nadia
2	2	Shiyi
3	3	Yizhe
4	4	Wei

Showing 1 to 4 of 4 entries

```
Data
mydat          4 obs. of 2 variables
Values
  name         chr [1:4] "Nadia" "Shiyi" "Yizh…
  student_num  num [1:4] 1 2 3 4
```

```
~/Dropbox (Personal)/Docs/sta130/lectures/lecture_4_new/
> View(mydat)
> student_num <- c(1, 2, 3, 4)
> name <- c("Nadia", "Shiyi", "Yizhe", "Wei")
> mydat <- data.frame(student_num, name)
> mydat
> View(mydat)
> student_num <- c(1, 2, 3, 4)
> name <- c("Nadia", "Shiyi", "Yizhe", "Wei")
> mydat <- data.frame(obsnum = student_num, student_name = name)
> mydat
> View(mydat)
> 
```

Name	Size	Mo
lect4_sta130_nt.Rmd	8.3 KB	Jan
rstudioscreenshot.png	272 KB	Jan
renvpane.png	267.4 KB	Jan
lect4_sta130_nt.html	3.1 MB	Jan
lect4_sta130_nt_cache		
lect4_sta130_nt_files		
rscript.png	174.2 KB	Jan
packages.png	218.7 KB	Jan
rstudioide.png	490.8 KB	Jan
coercion.png	113.8 KB	Jan
dataframe.png	124.2 KB	Jan
Week3_Testing101.Rmd	20 KB	Jan
dataframeview.png	432.5 KB	Jan

R Notation - [,]

- To extract a value or set of values from a data frame, write the data frame's name followed by a pair of square brackets with a comma [,].

```
mydat[ , ]
```

rows

Columns.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$A[i,j]$$

R Notation - [,]

```
mydat
```

```
##   obsnum student_name
## 1      1        Nadia
## 2      2        Shiyi
## 3      3        Yizhe
## 4      4          Wei
```

first row

```
mydat[1,2] # the value in row 1 and column 2
```

Second Column.

```
## [1] Nadia
## Levels: Nadia Shiyi Wei Yizhe
```

"Nadia"

```
mydat[c(1,2),2] # all values in rows 1 and 2 in second column
```

mydat[1,2]

mydat[c(1,1),2]

```
## [1] Nadia Shiyi
## Levels: Nadia Shiyi Wei Yizhe
```

Student_name is Stored as a factor.

R Notation - $

The `$` tells R to return all of the values in a column as a vector.

```
mydat$student_name
```

```
## [1] Nadia Shiyi Yizhe Wei
## Levels: Nadia Shiyi Wei Yizhe
```

```
vec <- mydat$student_name # assign it to vec
attributes(vec) # info associated with object vec
```

Save as a vector

```
## $levels
## [1] "Nadia" "Shiyi" "Wei"   "Yizhe"
##
## $class
## [1] "factor"
```

```
vec[2] # get second element of vector
```

then access 2nd element.

```
## [1] Shiyi
## Levels: Nadia Shiyi Wei Yizhe
```

R Notation - combine [,] and $

did not specify column.

```
mydat[mydat$obsnum == 1,] # first row of data frame and all columns
```

```
##   obsnum student_name
## 1      1        Nadia
```

```
mydat[mydat$obsnum == 1 | mydat$obsnum == 4 ,] # first and fourth rows of data frame and all columns
```

```
##   obsnum student_name
## 1      1        Nadia
## 4      4          Wei
```

or operator.

if obsnum = 1 or

obsnum = 4 then

TRUE

Missing Data - `NA`

- Missing information problems happen frequently in data science.
- For example a value is mising because the measurement was lost, corrupted, or never recorded.
- The `NA` character is a special symbol in R. It stands for “not available” and can be used as a placeholder for missing information.

```
1 + NA
```

```
## [1] NA
```

Missing Data - `na.rm()`

- Suppose you collected the ages of five students, but you forgot to record the fifth students age.

```
age <- c(19, 20, 17, 20, NA)
mean(age) # mean will be NA
```

5th element is missing.

```
## [1] NA
```

```
age <- c(19, 20, 17, 20, NA)
mean(age, na.rm = TRUE) # R will ignore missing values
```

```
## [1] 19
```

Identify and Set Missing Data - `is.na()`

is.na is TRUE if the value is missing (i.e., the value is NA) otherwise it is FALSE

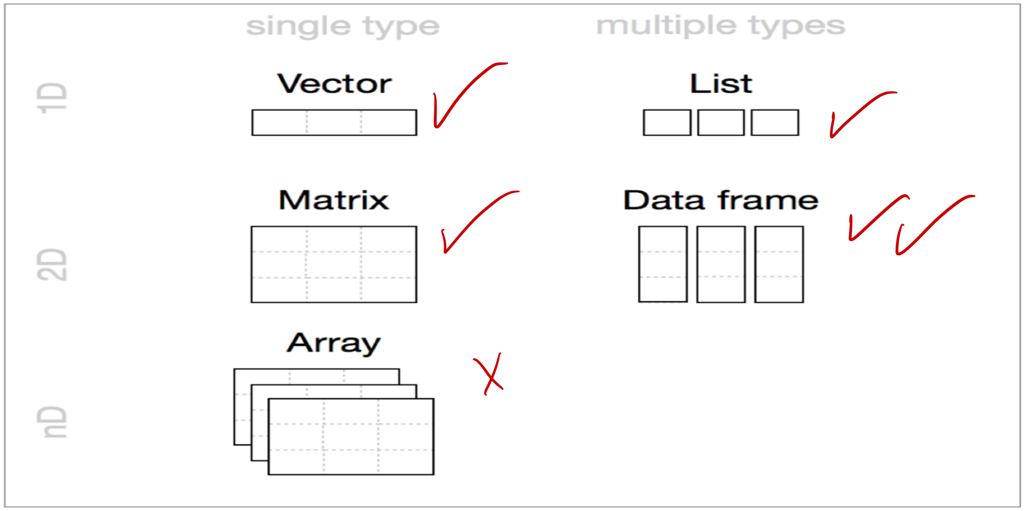
```
age <- c(19, 20, 17, 20, NA)
is.na(age) # check which elements of age are missing
```

```
## [1] FALSE FALSE FALSE FALSE  TRUE
```

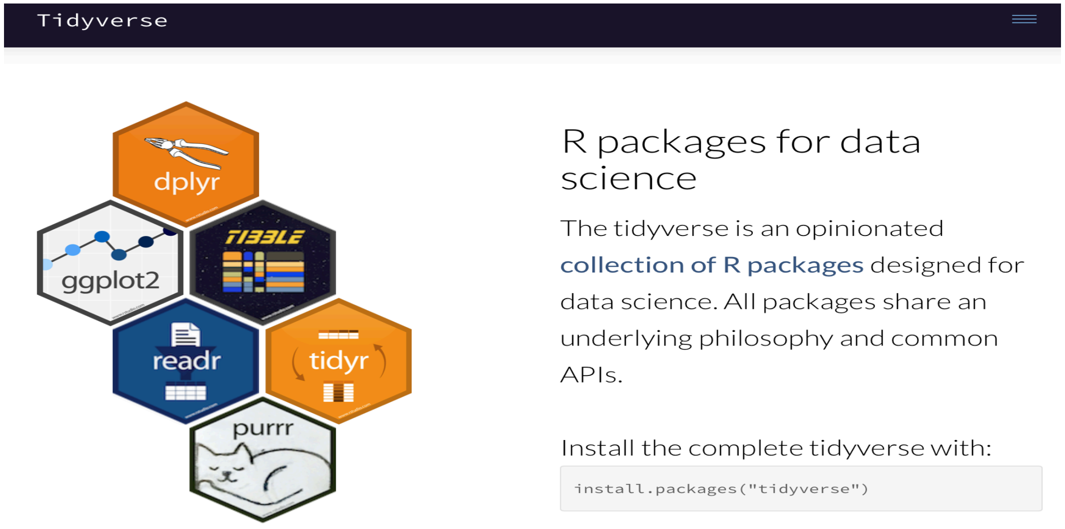
```
age[1] <- NA # set the first element of age to NA
age
```

```
## [1] NA 20 17 20 NA
```

Summary of R Data Structures



Tidyverse



R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

https://www.tidyverse.org

Canadian Flu Rates with `dplyr`

The provincial rates for the week ending January 6, 2018 are in the file fludat_prov.csv and the the size of the population in each province is in the file popdat.csv. The code below reads the files into R data frames.

```
library(tidyverse)
fludat_prov <- read_csv("fludat_prov.csv") # import data from file
popdat <- read_csv("popdat.csv") # import data from file
```

Canadian Flu Rates with **`dplyr`**

from dplyr lib

```
head(fludat_prov) # head shows the first six rows of a data frame
```

returns first 6 rows.

```
## # A tibble: 6 x 3
##   prov                      testpop_size  fluA
##   <chr>                            <int> <int>
## 1 Newfoundland                        96    12
## 2 Prince Edward Island                64    11
## 3 Nova Scotia                        144    23
## 4 New Brunswick                      347    80
## 5 Province of Québec                6361  1190
## 6 Province of Ontario               2320   344
```

# of people tested for flu

# of people + for flu A.

fluA / testpop_size

e.g., 12/96 in NFLD.

```
head(popdat)
```

```
## # A tibble: 6 x 3
##   prov                  prov_pop_size region
##   <chr>                         <int> <chr>
## 1 Nunavut                       35944 Territories
## 2 Alberta                     4067175 <NA>
## 3 Saskatchewan                1098352 West
## 4 Yukon                         35874 Territories
## 5 Manitoba                    1278365 West
## 6 British Columbia            4648055 West
```

Canadian Flu Rates with **`dplyr`**

How many Provinces/Territories are in the fludat_prov data frame?

```
fludat_prov %>% summarise(numprov = n()) # n() counts the number of rows in the data frame
```

returns the number of Rows in a dataframe.

```
## # A tibble: 1 x 1
##   numprov
##     <int>
## 1      13
```

Summarise(fludat_prov, numprov= n()).

Canadian Flu Rates with **`dplyr`**

Do any variables in fludat or popdat have missing values?

```
fludat_prov %>% filter(is.na(prov) == TRUE | is.na(testpop_size) == TRUE | is.na(fluA) == TRUE)
```

ō o no missing.

```
## # A tibble: 0 x 3
## # ... with 3 variables: prov <chr>, testpop_size <int>, fluA <int>
```

```
popdat %>% filter(is.na(prov) == TRUE | is.na(prov_pop_size) == TRUE | is.na(region) == TRUE)
```

```
## # A tibble: 2 x 3
##   prov    prov_pop_size region
##   <chr>           <int> <chr>
## 1 Alberta       4067175 <NA>
## 2 Quebec        8164361 <NA>
```

Canadian Flu Rates with `dplyr`

Recode specific values using R data frame notation [,] and $.

```
popdat$region[popdat$prov == "Alberta"] <- "West" #recode only the region value for Alberta
popdat$region[popdat$prov == "Quebec"] <- "East" #recode only the region value for Alberta
popdat$region #print region variable in popdat data
```

```
##  [1] "Territories" "West"        "West"        "Territories" "West"
##  [6] "West"        "East"        "East"        "Atlantic"    "Atlantic"
## [11] "Territories" "Atlantic"    "Atlantic"
```

Canadian Flu Rates with **dplyr** - Joining Two Tables with **inner_join()**

We can join two data frames with `inner_join(x,y)`: return all rows from x where there are matching values in y, and all columns from x and y. If there are multiple matches between x and y, all combination of the matches are returned.

```
fludat_prov %>% inner_join(popdat, by = "prov")
```

returns a data frame with 9 rows.

```
## # A tibble: 9 x 5
##   prov                      testpop_size  fluA prov_pop_size region
##   <chr>                            <int> <int>         <int> <chr>
## 1 Newfoundland                        96    12        519716 Atlantic
## 2 Prince Edward Island                64    11        142907 Atlantic
## 3 Nova Scotia                        144    23        923598 Atlantic
## 4 New Brunswick                      347    80        747101 Atlantic
## 5 Manitoba                           849   186       1278365 West
## 6 British Columbia                  1078   198       4648055 West
## 7 Yukon                               15     1         35874 Territories
## 8 Northwest Territories               28    10         41786 Territories
## 9 Nunavut                             18     1         35944 Territories
```

Why are there only 9 observations when there are 13 Provinces/Territories?

Canadian Flu Rates with **dplyr** - Joining Two Tables with **inner_join()**

```
fludat_prov$prov
```

```
##  [1] "Newfoundland"              "Prince Edward Island"
##  [3] "Nova Scotia"               "New Brunswick"
##  [5] "Province of Québec"        "Province of Ontario"
##  [7] "Manitoba"                  "Province of Saskatchewan"
##  [9] "Province of Alberta"       "British Columbia"
## [11] "Yukon"                     "Northwest Territories"
## [13] "Nunavut"
```

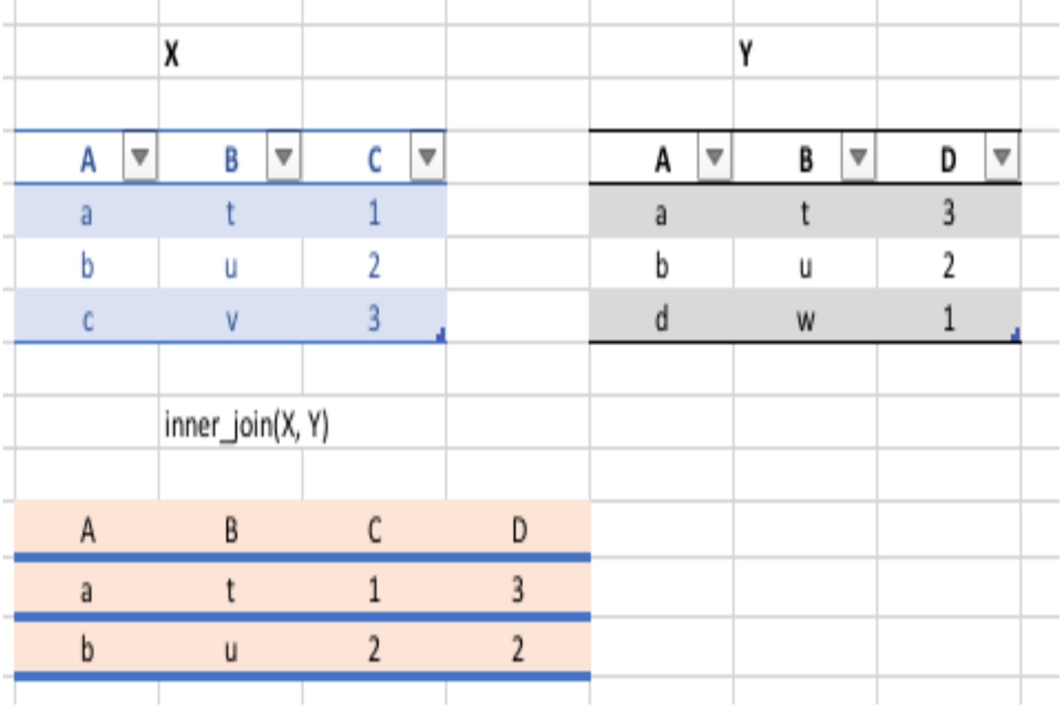
not the same

```
popdat$prov
```

```
##  [1] "Nunavut"               "Alberta"
##  [3] "Saskatchewan"          "Yukon"
##  [5] "Manitoba"              "British Columbia"
##  [7] "Ontario"               "Quebec"
##  [9] "Prince Edward Island"  "Newfoundland"
## [11] "Northwest Territories" "Nova Scotia"
## [13] "New Brunswick"
```

Province needs to be recoded. Exercise on this week's practice problems.

Canadian Flu Rates with **`dplyr`** - Joining Two Tables with **`inner_join()`**



X

A	B	C
a	t	1
b	u	2
c	v	3

Y

A	B	D
a	t	3
b	u	2
d	w	1

inner_join(X, Y)

A	B	C	D
a	t	1	3
b	u	2	2