STA130H1F

Class #2

Prof. Nathan Taback

2018-17-09

Welcome back to STA130 😀

Today's class

- Introduction to programming with R

Welcome back to STA130 😀

Today's class

- Introduction to programming with R
- Numerical descriptions of the distribution of quantitative variable

Tutorial grading

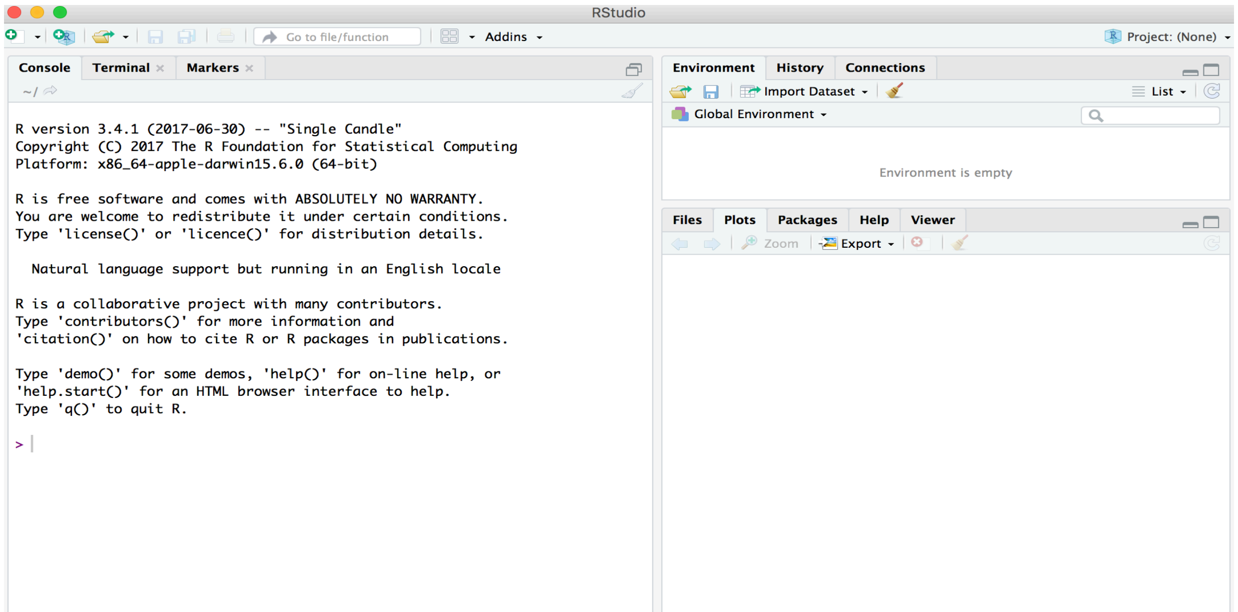
Tutorial grades will be assigned according to the following marking scheme.

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

Programming with R

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

RStudio User Interface



R Objects

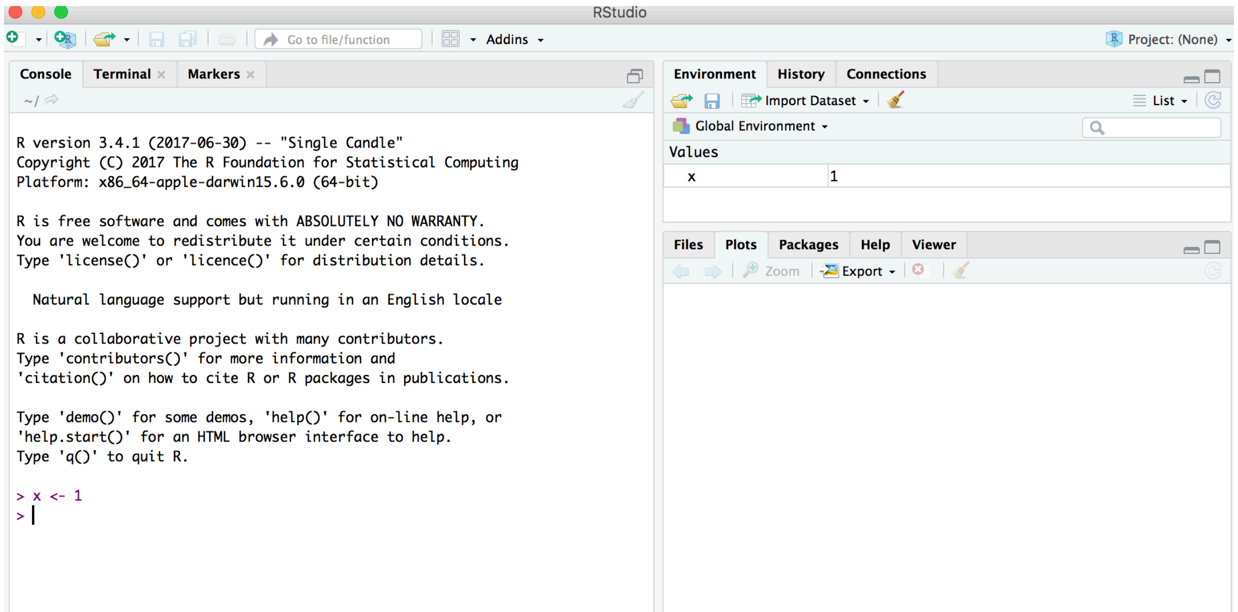
- R lets you save data by storing it inside an R object.
- What's an object? Just a name that you can use to call up stored data.

```
x <- 1
x
```

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

Environment Pane in RStudio

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



Functions

- R comes with many functions that you can use to do sophisticated tasks like random sampling.

Functions

- 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()`.

Functions

- 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
```

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

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

Function Constructor

- Every function in R has three basic parts: a name, a body of code, and a set of arguments.

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()`.

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() {
  add code here
}
```

Function Constructor

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] 7
```

Function Constructor

- We can create our own function with

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

Call the function `roll()`

```
roll()
```

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

NB: result will differ with every call

Function Arguments

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

```
roll2 <- function(numrolls) {
  die <- 1:6
  dice <- sample(die, size = numrolls, replace = TRUE)
  sum(dice)
}
```

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

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

```
roll2(10)
```

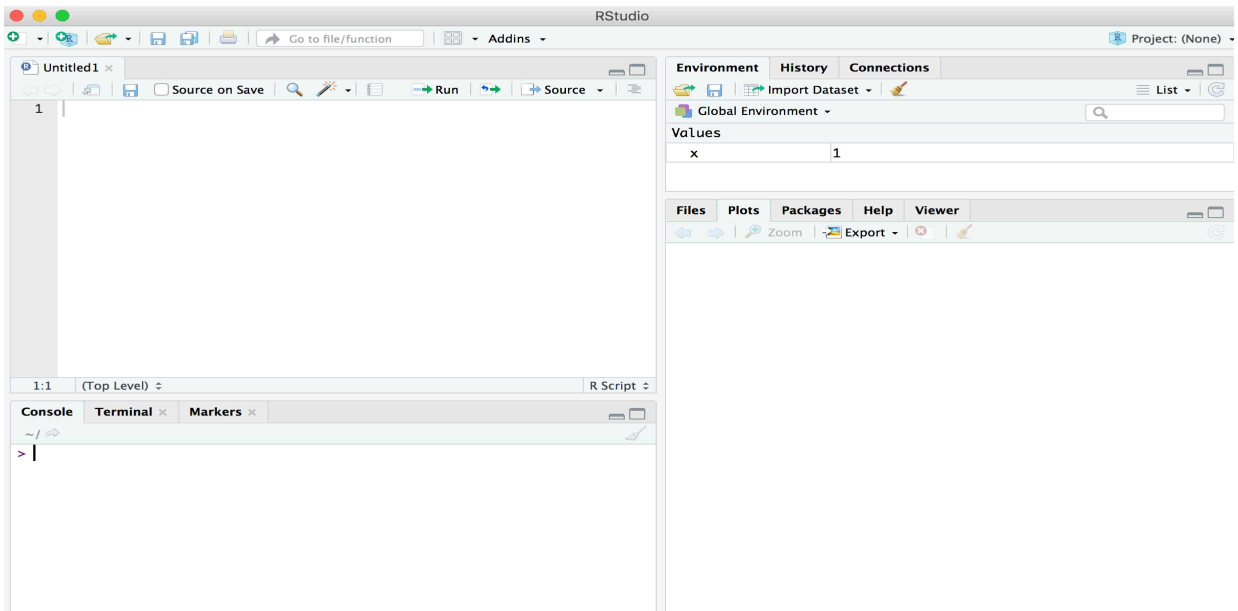
```
## [1] 38
```

```r
roll3 <- function(numrolls){
  die <- 1:6
  dice <- sample(die, size = numrolls, replace = F)
  sum(dice)
  }
```

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.

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.

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.

Packages

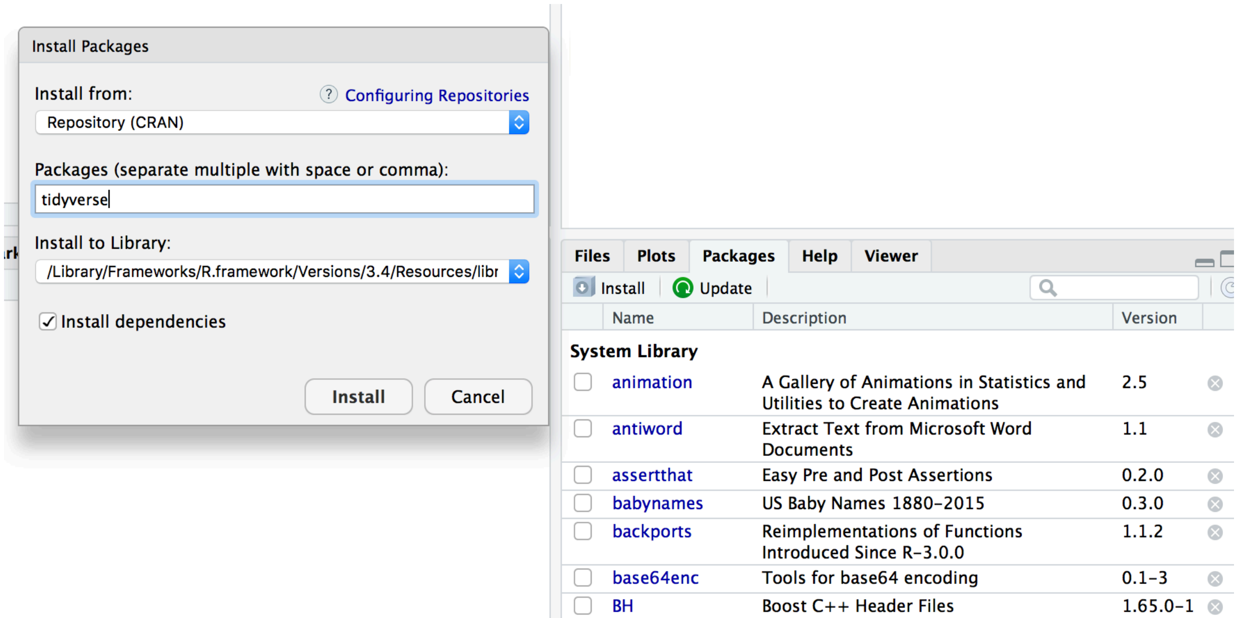
- You’re not the only person writing your own functions with R.
- Many professors, programmers, and statisticians use R to design tools that can help people analyze data.
- They then make these tools free for anyone to use.
- To use these tools, you just have to download them. They come as preassembled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

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.

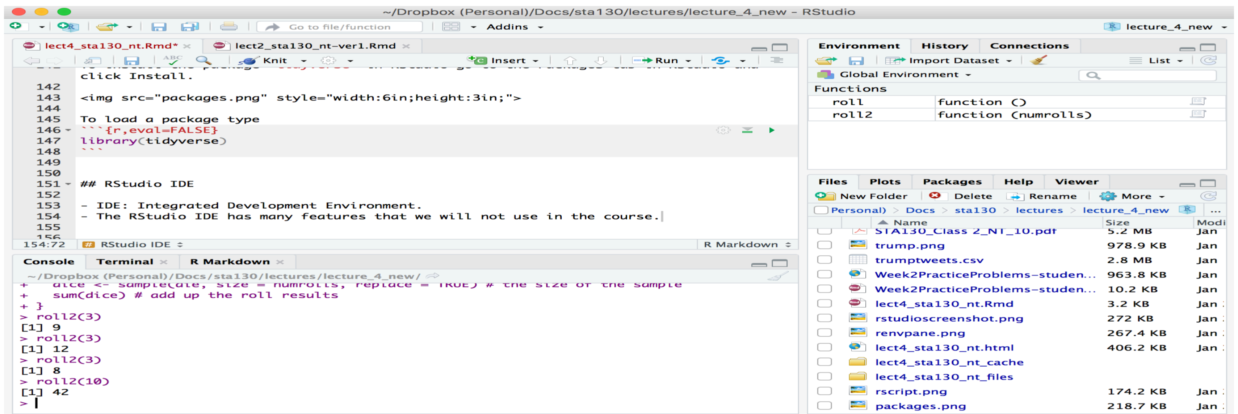


To load a package type

```
library(tidyverse)
```

RStudio IDE

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



- 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
```

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 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.

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

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.
- 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] 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 dimensions attribute with `dim()`.

```
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
```

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

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

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)
```

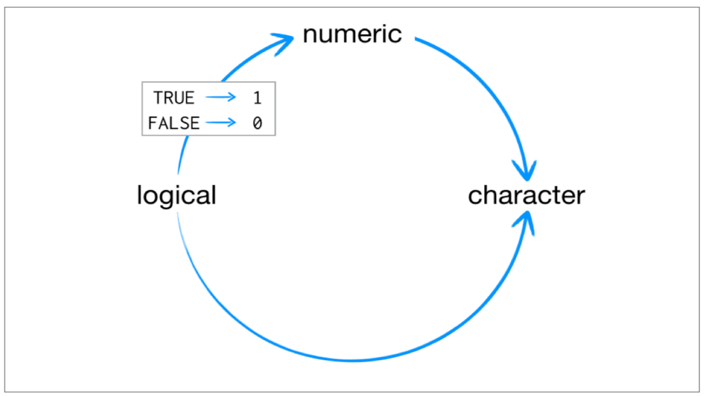
```
## [1] "integer"
```

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

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

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.



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

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

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

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 10 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.

```
list(1:10,
     "Prof. Taback",
     list(TRUE, FALSE))
```

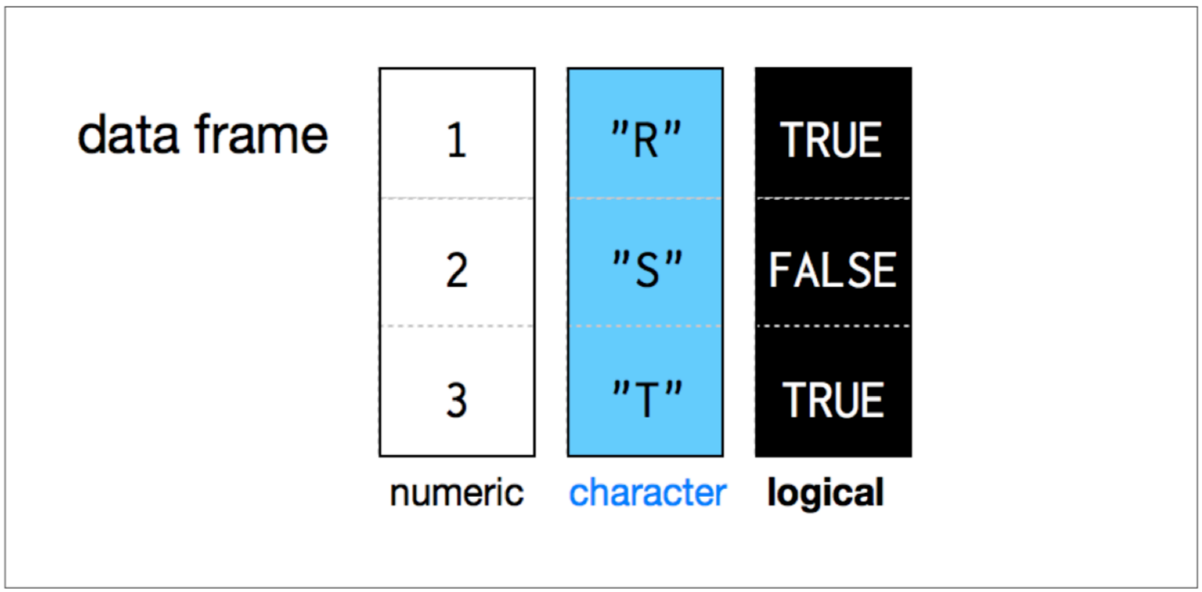
```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[2]]
## [1] "Prof. Taback"
##
## [[3]]
## [[3]][[1]]
## [1] TRUE
##
## [[3]][[2]]
## [1] FALSE
```

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 Frames

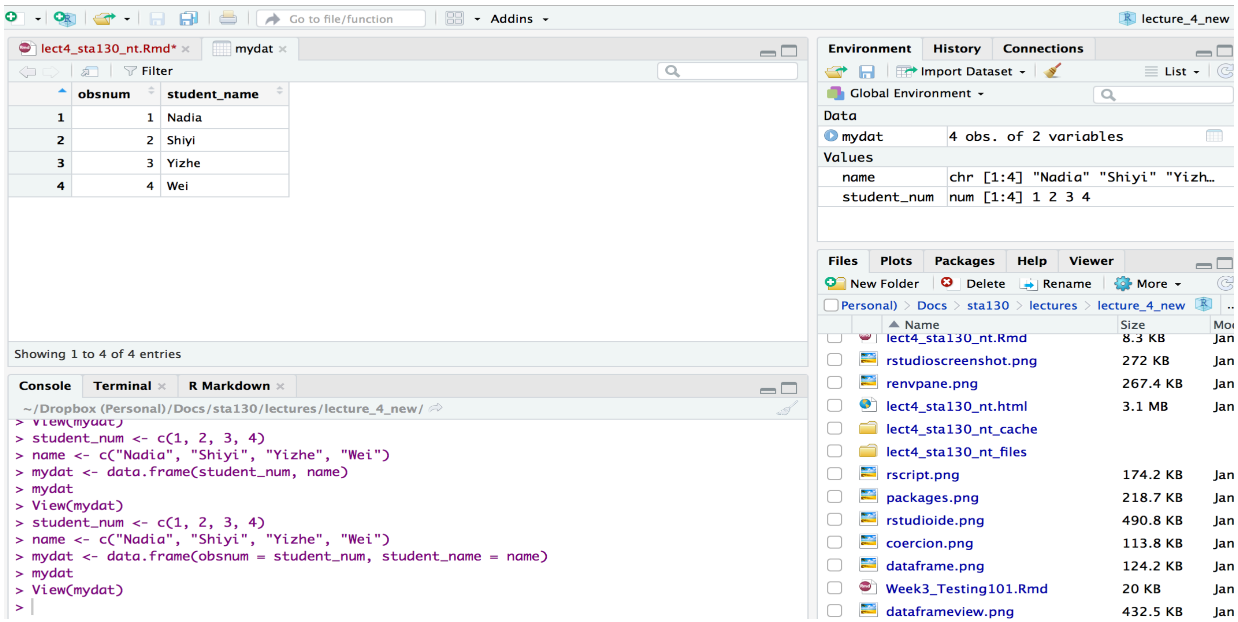
```
student_num <- c(1, 2, 3, 4)
name <- c("Nadia", "Shiyi", "Yizhe", "Wei")
mydat <- data.frame(obsnum = student_num, student_name = name)
mydat
```

```
##   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



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[ , ]
```

R Notation - [,]

```
mydat
```

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

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

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

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

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

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
```

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

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

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

R Notation - combine [,] and $

```
mydat
```

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

```
mydat[mydat$obsnum == 1,]
```

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

```
mydat[mydat$obsnum == 1 |
        mydat$obsnum == 4,]
```

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

```
mydat[mydat$obsnum != 1,]
```

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

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.

```r
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
```

```
## [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()`

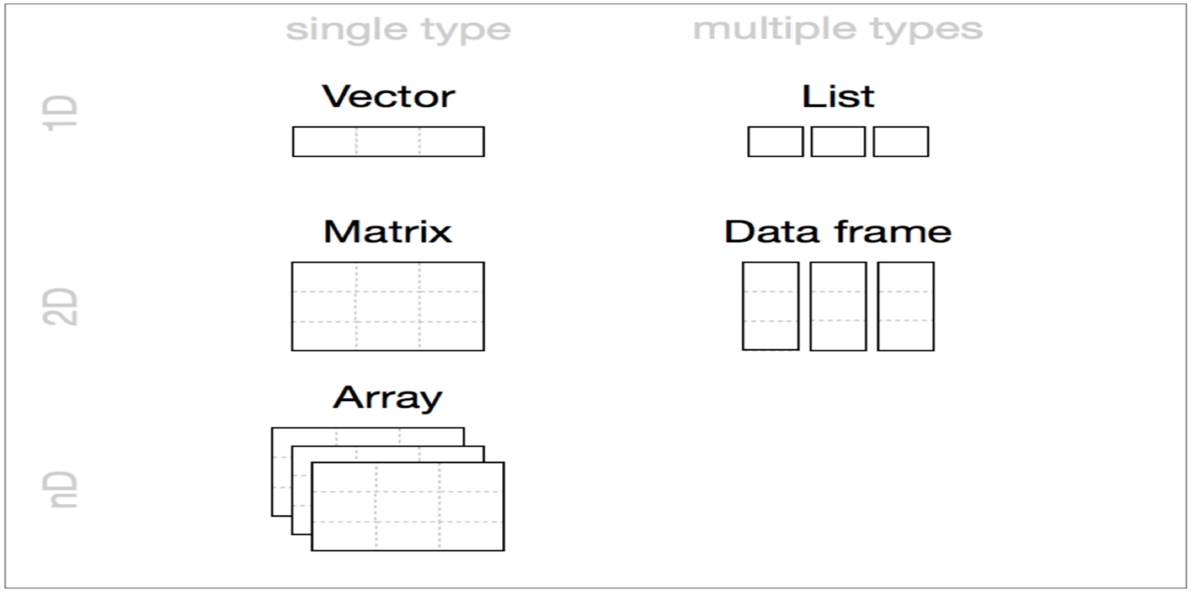
```
age <- c(19, 20, 17, 20, NA)
is.na(age) # check which elements of age are missing
```

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

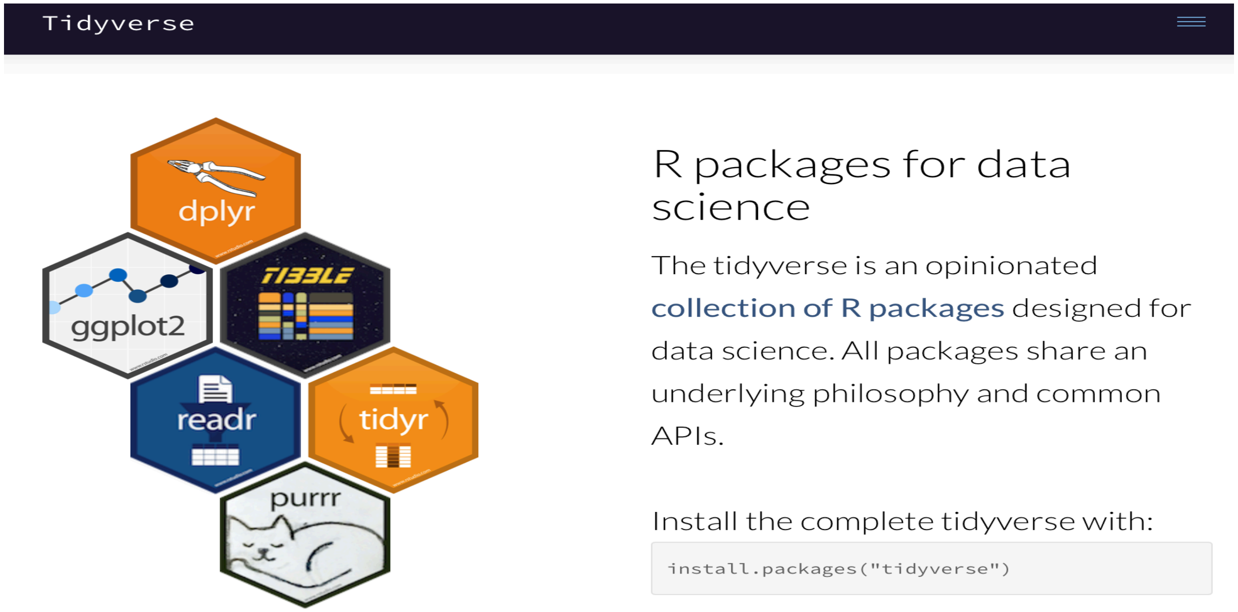
```
age[1] <- NA # set the first element of age to NA
age
```

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

Summary of R Data Structures



Tidyverse



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")
popdat <- read_csv("popdat.csv")
```

Canadian Flu Rates with dplyr

```
head(fludat_prov, n = 5)
```

```
## # A tibble: 5 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
```

```
head(popdat, n = 5)
```

```
## # A tibble: 5 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
```

Canadian Flu Rates with `dplyr`

How many Provinces/Territories are in the fludat_prov data frame? Use `summarise()` function in `dplyr`.

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

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

Canadian Flu Rates with `dplyr`

Do any variables in fludat or popdat have missing values?

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

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

```
filter(popdat,
       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 $.

```
 #recode region value for Alberta
popdat$region[popdat$prov == "Alberta"] <- "West"
 #recode region value for Quebec
popdat$region[popdat$prov == "Quebec"] <- "East"
popdat$region
```

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

Canadian Flu - Calculate Rate using `mutate()`

Transform existing variables to create a new variable using `mutate()`. The proportion of people testing positive in a province is

$$\text{Number with positive flu test}/\text{Number of people tested.}$$

```
fludat_prov1 <- mutate(fludat_prov, flu_prop = fluA/testpop_size)
head(fludat_prov1)
```

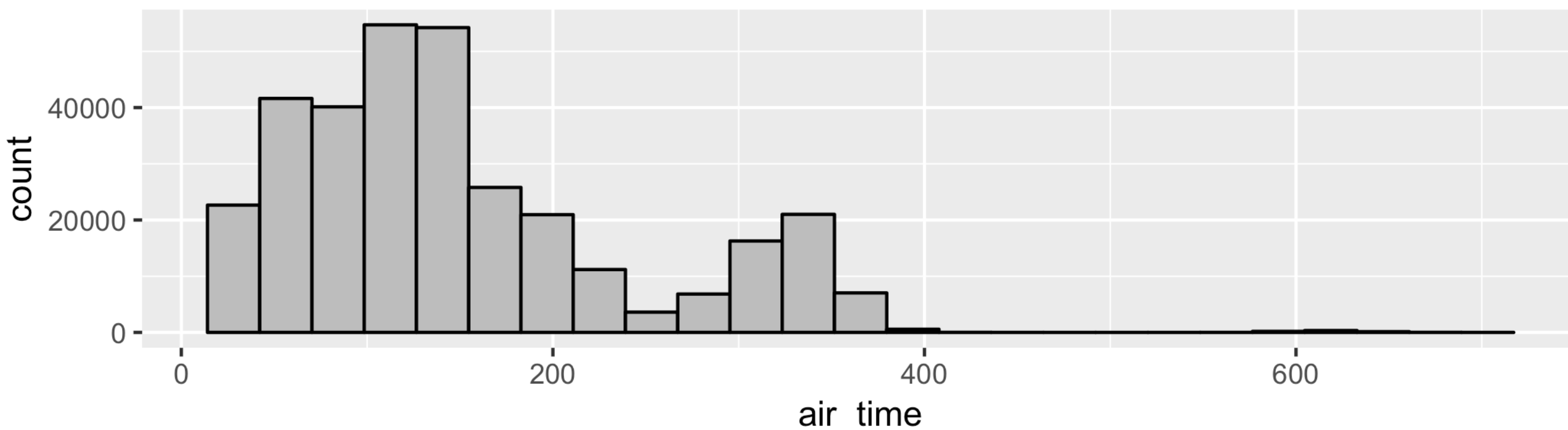
```
## # A tibble: 6 x 4
##   prov                      testpop_size  fluA flu_prop
##   <chr>                            <int> <int>    <dbl>
## 1 Newfoundland                        96    12    0.125
## 2 Prince Edward Island                64    11    0.172
## 3 Nova Scotia                        144    23    0.160
## 4 New Brunswick                      347    80    0.231
## 5 Province of Québec                6361  1190    0.187
## 6 Province of Ontario               2320   344    0.148
```

New York City Flights

- The R package `nyc13flights` contains contains information about all flights that departed from NYC (e.g. EWR, JFK and LGA) in 2013.
- The `flights` data set contains data on flights including the amount of time spent in the air.

```r
library(nycflights13)
summarise(flights,
          n = n(),
          time_ave = mean(air_time, na.rm = TRUE),
          time_sd = sd(air_time, na.rm = TRUE),
          time_med = median(air_time, na.rm = TRUE),
          time_25p = quantile(air_time, 0.25, na.rm = TRUE),
          time_75p = quantile(air_time, 0.75, na.rm = TRUE),
          time_iqr = IQR(air_time, na.rm = TRUE))
```

```
## # A tibble: 1 x 7
##        n time_ave time_sd time_med time_25p time_75p time_iqr
##    <int>    <dbl>   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 336776     151.    93.7      129       82      192      110
```



Numerical Summaries of the Distribution of a Quantitative Variable - Mean

The **mean** is a common way to measure the **center** of a distribution of data.

If $x_1, x_2, \ldots, x_n$ represent the n observed values then the mean is

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{\sum_{i=1}^{n} x_i}{n}.$$

Numerical Summaries of the Distribution of a Quantitative Variable - Variance

The **variance** is a common way to measure the **spread** of a distribution of data.

If $x_1, x_2, \dots, x_n$ represent the n observed values, and $\bar{x}$ the mean then the variance is

$$s^2 = \frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2}{n-1} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}.$$

The **standard deviation** is defined as the $s = \sqrt{s^2}$

The variance is roughly the average squared distance from the mean. The standard deviation is the square root of the variance and describes how close the data are to the mean.

Numerical Summaries of the Distribution of a Quantitative Variable - Quantiles

- The p^{th} quantile of a distribution is defined to be the value of the distribution x_p such that $p \times 100\%$ of the data are less than or equal to x_p.

Numerical Summaries of the Distribution of a Quantitative Variable - Quantiles

- The p^{th} quantile of a distribution is defined to be the value of the distribution x_p such that $p \times 100\%$ of the data are less than or equal to x_p.

- If $p = 0.50$ then $x_{0.5}$ is the value such that 50% of the data are less than $x_{0.5}$. This is also called the **median** or **second quartile**.

Numerical Summaries of the Distribution of a Quantitative Variable - Quantiles

- The p^{th} quantile of a distribution is defined to be the value of the distribution x_p such that $p \times 100\%$ of the data are less than or equal to x_p.
- If $p = 0.50$ then $x_{0.5}$ is the value such that 50% of the data are less than $x_{0.5}$. This is also called the **median** or **second quartile**.
- If $p = 0.25$ then $x_{0.25}$ is the value such that 25% of the data are less than $x_{0.25}$. This is also called the $\mathbf{25^{th}}$ **percentile** or **first quartile**.

Numerical Summaries of the Distribution of a Quantitative Variable - Quantiles

- The p^{th} quantile of a distribution is defined to be the value of the distribution x_p such that $p \times 100\%$ of the data are less than or equal to x_p.

- If $p = 0.50$ then $x_{0.5}$ is the value such that 50% of the data are less than $x_{0.5}$. This is also called the **median** or **second quartile**.

- If $p = 0.25$ then $x_{0.25}$ is the value such that 25% of the data are less than $x_{0.25}$. This is also called the $\mathbf{25^{th}}$ **percentile** or **first quartile**.

- If $p = 0.75$ then $x_{0.75}$ is the value such that 75% of the data are less than $x_{0.75}$. This is also called the $\mathbf{75^{th}}$ **percentile** or **third quartile**.