IE6600 Computation and Visualization for Analytics

BasicR

Zhenyuan Lu

(updated: 2021-09-15)

Image credit: RStudio

Image credit: Shiny, RStudio

Basic R

A sweet note

For your convenience, I have generated all the slides by RMarkdown in RStudio, which means all the original code syntax/chunk won't be messed up by different presentation programs or software.

Another sweet note

I won't use beamer style slides...which makes me headache and sleepy...

Beamer Style Slides as follows:

Tip: Update R language in Rstudio

For windows only:

```
install.pacages("installr")
library(installr)
updateR()
```

For mac: Go to R project

Packages needed

You'll also need to install some R packages. An R package is a collection of functions, data, and documentation that extends the capabilities of base R.

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

[1] Wickham, Hadley, and Garrett Grolemund. R For Data Science. OReilly, 2017.

Packages needed (cont'd)

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

This tells you that tidyverse is loading the `ggplot2`, `tibble`, `tidyr`, `readr`, `purrr`, `stringr`, `forcats`, and `dplyr` packages. These are considered to be the core of the tidyverse because you'll use them in almost every analysis.

[1] Wickham, Hadley, and Garrett Grolemund. R For Data Science. OReilly, 2017.

Function conflicts (cont'd)

```
library(tidyverse)
```

This also tells you that there are two functions from dplyr having conflicts with stats. You'll use `dplyr::` or `stats::` to specific the function from dplyr. This is a very common issue students may have.

[1] Wickham, Hadley, and Garrett Grolemund. R For Data Science. OReilly, 2017.

Installation of R Markdown

There are many other excellent packages that are not part of the tidyverse.

```
# Install r markdown from CRAN
install.packages("rmarkdown") # Conversion Tool
# Or if you want to test the development version,
# install from GitHub
if (!requireNamespace("devtools"))
  install.packages('devtools')
devtools::install_github('rstudio/rmarkdown')
```

[1] Wickham, Hadley, and Garrett Grolemund. R For Data Science. OReilly, 2017.

[2] Xie, Yihui, et al. R Markdown. CRC Press, 2019.

Installation of R Markdown (cont'd)

Be sure to also install TinyTex for those who have not installed LaTex before.

```
install.packages("tinytex")
tinytex::install_tinytex()  # install TinyTeX
```

If you have any issues with R Markdown, feel free to check: Q&A for IE6600

[1] Wickham, Hadley, and Garrett Grolemund. R For Data Science. OReilly, 2017.

[2] Xie, Yihui, et al. R Markdown. CRC Press, 2019.

Packages needed (cont'd)

```
install.packages("ggplot2") # Visualization Tool, install "tidyverse" instead
install.packages("nycflights13") #Airline Flights data
install.packages("gapminder") #world development data
```

or if you would like to make your own packages for supporting HUMAN BEINGS DEVELOPMENT by learning R Packages[2]

[1] Wickham, Hadley, and Garrett Grolemund. R For Data Science. OReilly, 2017.

[2] Wickham, Hadley. R Packages. OReilly Media, 2015.

Tip: Check if packages installed

```
libs <- c("ggplot2", "nycflights13", "gapminder")

x <- sapply(libs, function(x)
  if (!require(
    x,
    character.only = T,
    warn.conflicts = F,
    quietly = T
  ))
    install.packages(x))
```

References

R Programming

- R for Data Science (open license)
- Cookbook for R (open license)
- Text Mining with R (open license)
- R for Everyone (library access)

Documentation

- R Markdown (open license)

R Visualization

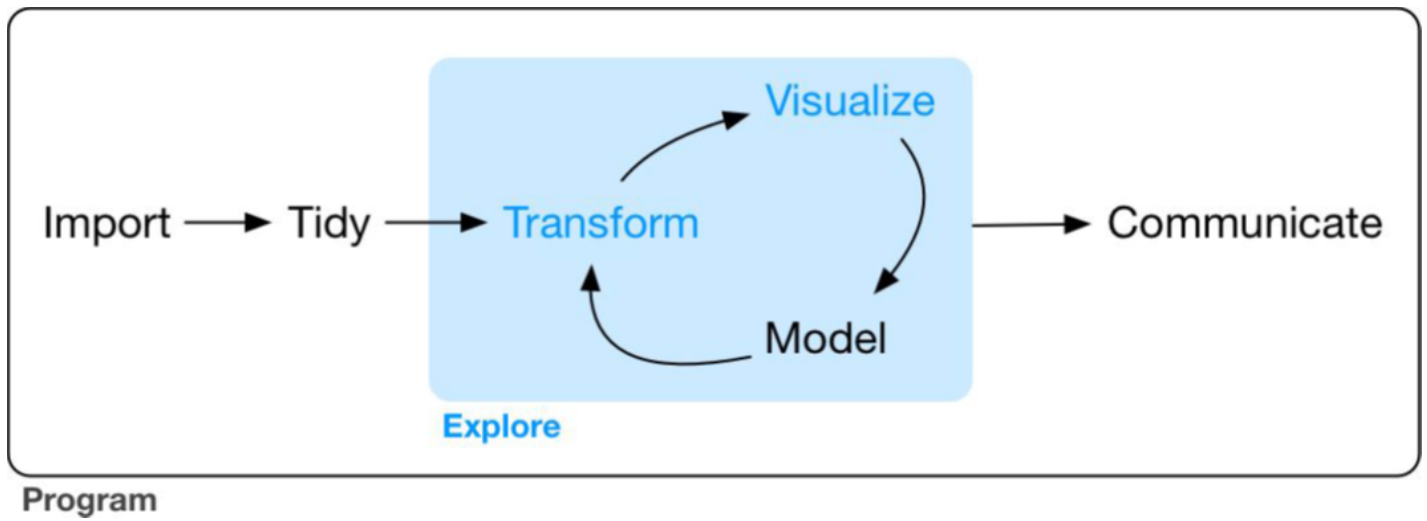
- ggplot2 for static Viz.
- plotly for interactive Viz.
- Shiny for web app

Additional

- Advanced R (open license)
- R Packages (open license)
- R Cheatsheets List(very useful)

Goal

The goal of the class IE6600 is to give you a solid foundation for most of the common data science tools in R and Tableau, and for the preparation to the advanced modeling, machine learning, and other data science fields. We will follow the following model for the rest of the semester.



Basic Math

R is a powerful tool for all manner of calculation since it is developed up by statisticians not programmers, which also means it is very friendly for data manipulation and scientific computations.

```
# 1 plus 1 *
1+1
```

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

```
# Another math
1/(1+1)
```

```
## [1] 0.5
```

[*]Please keep a good habit of writing down the descriptions/comments before each essential script, although we know most of you don't like doing it.

Functions in R

Function is an object. R has a large number of in-built and third-party functions.

Example Let's draw a lottery from 10 numbers. If it's one of 1 to 9 numbers, which means my students love me.

```
if (sample(10, 1)%in%c(1:9)) {
  print("My students love me")
} else{
  "They hate me"
}
```

```
## [1] "My students love me"
```

We will mention how to build up a function in next chapter

Variables

- Variables are an integral part of any programming language and R offers a great deal of flexibility.
- Unlike statically typed languages such as C++, R does not require variable types to be declared. A variable can take on any available data type.
- It can also hold any R object such as a function, the result of an analysis or a plot. A single variable can at one point hold a number, then later hold a character and then later a number again.

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Variables Assignments

The valid assignment operators are <- and = with the first being preferred

```
x <- 13 # Recommended
x
```

```
## [1] 13
```

```
x = 13
x
```

```
## [1] 13
```

[1] R is also friendly for who is familiar with SQL where a single equal sign(=) tests for equality

[2] Recommended to use <- instead of =

Variables Assignments (cont'd)

```
assign("x", 13)
x
```

```
## [1] 13
```

[1] R is also friendly for who is familiar with SQL where a single equal sign(=) tests for equality

[2] Recommended to use <- instead of =

Removing Variables

`rm()` function for removing variables

```
#Store 13 to the variable x
x <- 13
```

```
#Removing x
rm(x)
#The x is gone
x
```

```
## Error in eval(expr, envir, enclos): object 'x' not foun
```

Although most of the programmers are intended to ignore "warning message" when coding, they always pay attention on the "Error" alert.

Variables, Case Sensitive

- Variable names are case sensitive, which can trip up people coming from a language like SQL or Visual Basic.

```
#Save 1 to variables
variables <- 1
variables
```

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

```
#not for Variables due to the case sensitive
Variables
```

```
## Error in eval(expr, envir, enclos): object 'Variables' not found
```

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Data Types

There are numerous data types in R that store various kinds of data. The four main types of data most likely to be used are

- `numeric` (double or integer, most of the time R won't differentiate between integer or double)
- `character (string)`
- `Date (time-based)`
- `logical (TRUE/FALSE)`

Show data set in R

```
data()
```

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Data Types (cont'd)

If you want to check the type of data contained in a variables, class function can be used for that.

```
x <- 13
#Check the data type of x
class(x)
```

```
## [1] "numeric"
```

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Numeric

As expected, R excels at running numbers, so numeric data is the most common type in R. The most commonly used numeric data is numeric. This is similar to a float or double in other languages.

Testing whether a variable is numeric is done with the function is.numeric.

```
x <- 13
#Testing if x is numeric
is.numeric(x)
```

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

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Numeric (cont'd)

To store a integer to the variable, be sure to add "L" after the numeric data.

```
#Save 3 to the x
x <- 3
is.integer(x)
```

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

```
#Save 3 as an integer to the x
x <-3L
is.integer(x)
```

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

```
is.numeric(x)
```

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

Character

R has two primary ways of handling character data: character and factor.

```
# Store "RStudio" to x as character data
x <- "RStudio"
x
```

```
## [1] "RStudio"
```

```
# Store "RStudio" to x as factor data by using factor function*
x <- factor("RStudio")
x <- factor(c("a", "c","D"))
x <- factor(c(1,2,3))
x
```

```
## [1] 1 2 3
## Levels: 1 2 3
```

Character - length

```
# Find the length of variable x
x <- "RStudio"
nchar(x)
```

```
## [1] 7
```

```
# nchar can only be used for character and numeric data
x <- factor("RStudio")
nchar(x)
```

```
## Error in nchar(x): 'nchar()' requires a character vector
```

```
# Find the length of numeric variable y
y <- 345
nchar(y)
```

```
## [1] 3
```

Date

Dealing with dates and times can be difficult in any language, and to further complicate matters R has numerous different types of dates. The most useful are Date and POSIXct. Date stores just a date while POSIXct stores a date and time. Both objects are actually represented as the number of days (Date) or seconds (POSIXct) since January 1, 1970.

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Date (cont'd)

How to save date data

```
#Store "2020-01-15" as date
date <- as.Date("2020-01-15")
as.numeric(date)
```

```
## [1] 18276
```

```
#Test the data type of date
class(date)
```

```
## [1] "Date"
```

```
#Convert the date to numeric
as.numeric(date)
```

```
## [1] 18276
```

Question

```
as.numeric(2020-01-15)
```

Data Wrangling (we will mention this chapter later)

```
install.packages("stringr") # for string
install.packages("lubridate") # for date
install.packages("forcats") # for factor
```

Wickham, Hadley, and Garrett Grolemund. R For Data Science. OReilly, 2017.

Logical

Logical is a way of representing data that can be either TRUE or FALSE.

```
x <- 13
is.numeric(x)
```

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

```
x1 <- "I am awesome"
is.numeric(x1)
```

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

```
is.character(x1)
```

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

```
is.na(x1)
```

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

Logical (cont'd)

```
# TRUE/FALSE

x <- TRUE
class(x)
```

```
## [1] "logical"
```

```
is.logical(x)
```

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

Logical (cont'd)

Numerically, TRUE is the same as 1 and FALSE is the same as 0.

```
TRUE * 5
```

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

```
c(TRUE, TRUE, TRUE)*1
```

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

```
FALSE * 5
```

```
## [1] 0
```

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Logical (cont'd)

Logical also can do the comparison between two numbers, or characters

```
# does 1 equal 1?
1 == 1
```

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

```
# does 1 not equal 1?
1 != 1
```

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

```
# is 1 less than 1?
1 < 1
```

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

```
# is 1 less than or equal to 2?
1 <= 2
```

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

```
# is the length of
# 'i' equal to 'IE6600'?
"i" == "IE6600"
```

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

```
# is ' i' less than 'IE6600'?
"i" < "IE6600"
```

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

Vector

- A vector is `a collection of elements, all of the same type.` For instance, c(1, 3, 2, 1, 5) is a vector consisting of the numbers 1, 3, 2, 1, 5, in that order. Similarly, c("R", "Excel", "SAS", "Excel") is a vector of the character elements "R," "Excel," "SAS" and "Excel." `A vector cannot be of mixed type.`

- Vectors do not have a dimension, meaning there is no such thing as a column vector or row vector. These vectors are not like the mathematical vector where there is a difference between row and column orientation.

- Column or row vectors can be represented as one-dimensional matrices, we will discuss this in the next section of advanced data structures.

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Vector (cont'd)

Using `c()` function is the most common way to create a vector

```
x <- c("IE6600", "Data", "Visualization")
x
```

```
## [1] "IE6600"        "Data"          "Visualization"
```

```
y <- c(2,3)
y
```

```
## [1] 2 3
```

```
class(y)
```

```
## [1] "numeric"
```

Vector - operations

Now we have a vector contains 1 to 5 numbers, there are two ways to do this:

Using `c()` function is the most common way to create a vector

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

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

```
y <- c(1:5)
y
```

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

You may also want to do some manipulation for your vectors

```
# Multiply each number by 3
x * 3
```

```
## [1]  3  6  9 12 15
```

```
# subtraction, division,
# or exponentiation are the same
x^2
```

```
## [1]  1  4  9 16 25
```

Vector - operation (cont'd)

```
# Squared root
sqrt(x)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

Vector - operation (cont'd)

Vector operations can be extended even further. Let's say we have two vectors of equal length. Each of the corresponding elements can be operated on together.

Using `c()` function is the most common way to create a vector

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

```
y <- c(1:5)
```

Be sure to check the length of x and y, which needs to be the same

```
length(x)
```

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

```
length(y)
```

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

Vector - operation (cont'd)

```
# Add x to y
x + y
```

```
## [1]  2  4  6  8 10
```

```
# Multiply x by y
x * y
```

```
## [1]  1  4  9 16 25
```

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Vector - operation (cont'd)

```
# Divided by y
x / y
```

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

```
# x to the power of y
x^y
```

```
## [1]    1    4   27  256 3125
```

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Vector - operation (cont'd)

What if these two vectors are not in the same length, what will happen?

Question

```
x <- c(1:5)

x + c(1,2)
???
```

Here is the answer

```
x <- c(1:5)

x + c(1,2)
```

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

HOW?

The shorter vector will have a loop of adding up to the longer vector 1+1, 2+2, 3+1, 4+2, 5+1 Thus

```
x + c(1,2)
[1] 2 4 4 6 6
```

Vector - comparisons

Comparisons also work on vectors. Here the result is a vector of the same length containing TRUE or FALSE for each element.

```
x <- c(1:5)
x
```

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

```
# if each of the element in x are smaller and equal to 3?
x <=3
```

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

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Vector - operation (cont'd)

How to count the number and length of the elements within vector

```
x <- c(1:5)
# Check how many characters of a vector
length(x)
```

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

```
# Check how many characters of each element
nchar(x)
```

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

Vector - Operation (cont'd)

Name your elements

```
c("I", "am", "Handsome")
```

```
## [1] "I"        "am"       "Handsome"
```

```
c(One="I", Two="am", Three="Handsome")
```

```
##        One        Two      Three
##        "I"       "am" "Handsome"
```

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Vector - operation (cont'd)

Accessing individual elements of a vector by using square bracket [] and put the number to choose which elements of a vector by order

```
x <- c(1:5)
x
```

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

```
#Access the 2nd element of the vector x
x[2]
```

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

```
#Access the 1st and 3rd elements of x
x[c(1,3)]
```

```
## [1] 1 3
```

```
#Access the 1st to 3rd elements of x
x[1:3]
```

```
## [1] 1 2 3
```

Factor

Factors are an important concept in R, especially when building models.
Sometimes users may have some errors due to the incorrectly using of factors.
We will mention more in the ggplot2 session

Factor (cont'd)

We can create and convert to factor by using factor and as.factor function, respectively

```
#Save new elements to x
x <- c("I", "am", "awesome")
x
```

```
## [1] "I"       "am"      "awesome"
```

The levels of a factor are the unique values of the factor variables. Here, the default levels are based on the alphabetical order of "am", "awesome", and "I".

```
#Converting x to factor
x <- as.factor(x)
x
```

```
## [1] I       am      awesome
## Levels: am awesome I
```

Factor (cont'd)

We can change the levels

```
x <- factor(x, levels=c("I","awesome","am"))
x
```

```
## [1] I       am      awesome
## Levels: I awesome am
```

Missing Data

Missing data plays a critical role in both statistics and computing, and R has two types of missing data, NA and NULL. While they are similar, they behave differently and that difference needs attention.

`We will talk about more details later in this semester`

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

NA

NA will often be seen as just another element of a vector. is.na tests each element of a vector for missingness.

```
#Save numbers included NA to x
x <- c(1, 2, 3, NA, 5)
x
```

```
## [1]  1  2  3 NA  5
```

```
#Check how many elements within x
length(x)
```

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

```
#Check if NA
is.na(x)
```

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

NULL

- NULL is the absence of anything. It is not exactly missingness, it is nothingness. Functions can sometimes return NULL and their arguments can be NULL.
- NULL is often returned by expressions and functions whose value is undefined.
- An important difference between NA and NULL is that NULL is atomical and cannot exist within a vector. If used inside a vector it simply disappears

The NULL Object, R Documentation

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

NULL (cont'd)

```
#Save numbers included NULL to x
x <- c(1, 2, 3, NULL, 5)
x
```

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

```
#Check how many elements within x
length(x)
```

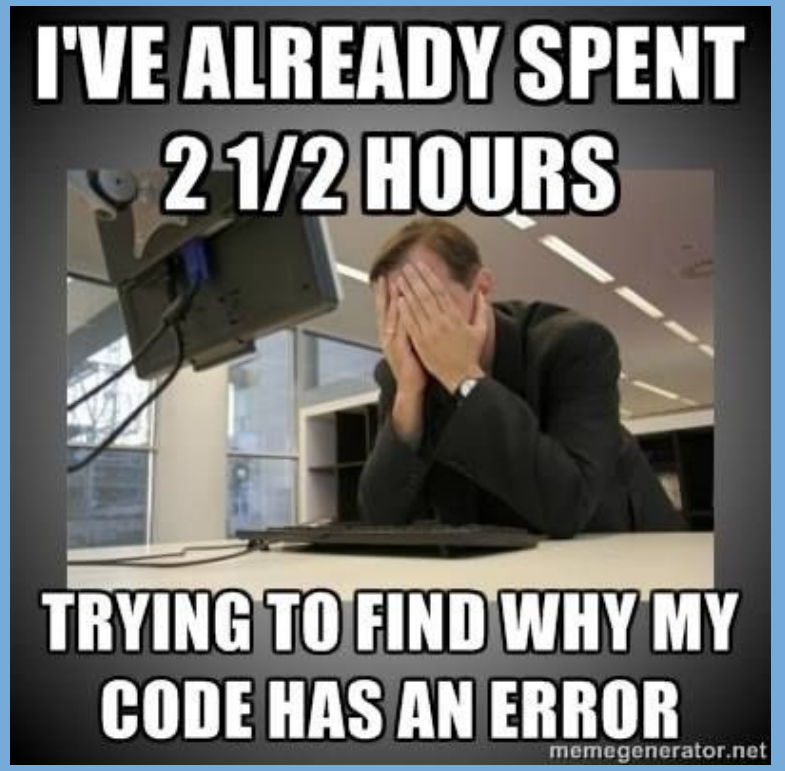
```
## [1] 4
```

```
#Check if all the elements within x are NULL
is.null(x)
```

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

Tips

If you are not sure about what will happen after running one line of script/code/anything, just type in and see what will happen. You will be surprised.



Calling Functions

```
# Do average of 1-3
mean(c(1,2,3))
```

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

```
# Do sum of 1-3
sum(c(1,2,3))
```

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

Pipe

A new paradigm for calling functions in R is the pipe %>% *(ctrl+shift+M for Win, cmd+shit+M for mac)*. The pipe works by taking the value or object on the left-hand side of the pipe and inserting it into the first argument of the function that is on the right-hand side of the pipe.

```
y <- c(1,2,3)
mean(y)
```

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

```
c(1,2,3) %>% mean
```

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

Life is short, use pipe **%>%**

Advanced Data Structure

The most common are the `data frame`, `matrix` and `list`. Of these, the `data frame` will be the most familiar to anyone who has used a spreadsheet, the matrix to people familiar with matrix math and the list to programmers.

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Data frames

Perhaps one of the most useful features of R is the data.frame. It is one of the most often cited reasons for R's ease of use. On the surface a data.frame is just like an `Excel spreadsheet` in that it has `columns` and `rows`. In statistical terms, each column is a variable and each row is an observation.

In terms of how R organizes `data.frames`, `each column` is actually `a vector`, each of which has the same length. This also implies that `within a column each element must be of the same type`, just like with vectors.

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Data frames

There are numerous ways to construct a data.frame, the simplest being to use the data.frame function

Let's create a basic data.frame using some of the vectors we have already introduced, namely x, y and a.

```
x <- 1:5
y <- -1:3
a <- c("I","am","an","awesome","student")
#Save x, y, a to xya.df as data frame in columns
xya.df <- data.frame(x,y,a)
xya.df
```

```
##   x  y       a
## 1 1 -1       I
## 2 2  0      am
## 3 3  1      an
## 4 4  2 awesome
## 5 5  3 student
```

Data frames

The second way to create a data.frame

```
xya.df <- data.frame(1:5,-1:3,c("I","am","an","awesome","student"))
xya.df
```

```
##   X1.5 X.1.3 c..I....am....an....awesome....student..
## 1    1    -1                                       I
## 2    2     0                                      am
## 3    3     1                                      an
## 4    4     2                                 awesome
## 5    5     3                                 student
```

Not recommend to use the same name as the default dataset's, e.g. diamonds, etc.

Data frames: change column names

```
xya.df <- data.frame(first=x, second=y, third=a)
xya.df
```

```
##   first second   third
## 1     1     -1       I
## 2     2      0      am
## 3     3      1      an
## 4     4      2 awesome
## 5     5      3 student
```

```
#or
names(xya.df) <- c(x="first", y="second", a="third")
xya.df
```

```
##   first second   third
## 1     1     -1       I
## 2     2      0      am
## 3     3      1      an
## 4     4      2 awesome
## 5     5      3 student
```

Data frames: rows and columns

Data.frames are complex objects with many attributes. The most frequently checked attributes are the number of rows and columns.

```
xya.df
```

```
##   first second   third
## 1     1     -1       I
## 2     2      0      am
## 3     3      1      an
## 4     4      2 awesome
## 5     5      3 student
```

```
#Check the number of rows
nrow(xya.df)
```

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

```
#Check the number of columns
ncol(xya.df)
```

```
## [1] 3
```

```
#Check the dimension of the data frames
dim(xya.df)
```

```
## [1] 5 3
```

```
#Check the column names
names(xya.df)
```

```
## [1] "first"  "second" "third"
```

Question

How to check the third columns names of xya.df

```
## [1] "third"
```

Data frames: rows and columns (cont'd)

Usually a data.frame has far too many rows to print them all to the screen, so thankfully the head function prints out only the first few rows

```
xya.df
```

```
##   first second   third
## 1     1     -1       I
## 2     2      0      am
## 3     3      1      an
## 4     4      2 awesome
## 5     5      3 student
```

```
#Check the first few rows
head(xya.df)
```

```
##   first second   third
## 1     1     -1       I
## 2     2      0      am
## 3     3      1      an
## 4     4      2 awesome
## 5     5      3 student
```

```
#Check the first three rows
head(xya.df, n=3)
```

```
##   first second third
## 1     1     -1     I
## 2     2      0    am
```

$ operator

Since each column of the data.frame is an individual vector, it can be accessed individually and each has its own class. Like many other aspects of R, there are multiple ways to access an individual column. There is the $ operator and also the square brackets.

```
xya.df
```

```
##   first second   third
## 1     1     -1       I
## 2     2      0      am
## 3     3      1      an
## 4     4      2 awesome
## 5     5      3 student
```

```
#Access the "second" column
xya.df$second
```

```
## [1] -1  0  1  2  3
```

Add new column to data frame

```
xya.df$newColumn <- c(1:5)
```

[] square bracket

```
xya.df
```

```
##   first second   third newColumn
## 1     1     -1       I         1
## 2     2      0      am         2
## 3     3      1      an         3
## 4     4      2 awesome         4
## 5     5      3 student         5
```

```
#Access the 5th row and 2nd column
xya.df[5,2]
```

```
## [1] 3
```

```
#Access the 5th row
xya.df[5,]
```

```
##   first second   third newColumn
## 5     5      3 student         5
```

```
#Access the 1st and 2nd row; and 3rd column
xya.df[c(1,2),3]
```

```
## [1] "I"  "am"
```

Data type of selected column

All of these methods have differing outputs. Some return a vector, some return a single-column data.frame. To ensure a single-column data.frame while using single-square brackets, there is a third argument: `drop=FALSE`. This also works when specifying a single column by number.

```
xya.df[, "third"]
```

```
## [1] "I"       "am"      "an"      "awesome" "student"
```

```
class(xya.df[, "third"])
```

```
## [1] "character"
```

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Data type of selected column (cont'd)

```
xya.df[, "third", drop=FALSE]
```

```
##     third
## 1       I
## 2      am
## 3      an
## 4 awesome
## 5 student
```

```
class(xya.df[, "third",drop=FALSE])
```

```
## [1] "data.frame"
```

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Data type of selected rows (cont'd)

```
xya.df[xya.df$second>1, ]
```

```
##   first second   third newColumn
## 4     4      2 awesome         4
## 5     5      3 student         5
```

```
xya.df[xya.df$second>1,2:3]
```

```
##   second   third
## 4      2 awesome
## 5      3 student
```

Lists

A list can contain all numeric or characters or a mix of the two or data.frames or, recursively, other lists.

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Create list

A list can contain all numeric or characters or a mix of the two or data.frames or, recursively, other lists.

Lists are created with the list function where each argument to the function becomes an element of the list.

```
#Create a three element list
list(1,2,3)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
```

```
#Create a single element list
list(c(1,2,3))
```

```
## [[1]]
## [1] 1 2 3
```

Create list (cont'd)

```
#Create a two element list
list(c(1,2,3), 1:4)
```

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

```
#Create a two element list, one element is a factor, the other one is a 3 element vectors
b <-  c("master", "PhD", "undergrade", "others", "master")
list(c(1:3),b)
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "master"     "PhD"        "undergrade" "others"     "master"
```

Rename lists

Using `names()` function to name the list

```
#Create a two element list, one element is a factor, the other one is a 3 element vectors
d <- list(c(1:3),b)

#Name the 1st list of d, number, and 2nd with degree.
names(d) <- c("number", "degree")
d
```

```
## $number
## [1] 1 2 3
##
## $degree
## [1] "master"     "PhD"        "undergrade" "others"     "master"
```

Rename lists (cont'd)

Names can also be assigned to list elements during creation using name-value pairs.

```
list1 <- list(number=1:3, degree=b)
list1
```

```
## $number
## [1] 1 2 3
##
## $degree
## [1] "master"     "PhD"        "undergrade" "others"     "master"
```

Access a specific list

To access an individual element of a list, use double square brackets, specifying either the element number or name. Note that this allows access to only one element at a time.

```
list1 <- list(number=1:3, degree=b)

#Access the first list
list1[[1]]
```

```
## [1] 1 2 3
```

```
list1[1]
```

```
## $number
## [1] 1 2 3
```

Add one more list to an exist list

It is possible to append elements to a list simply by using an index (either numeric or named) that does not exist.

```
d <- list(number=1:3, degree=b)

#Check how many elements within one list d
length(d)
```

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

```
#Add one list of four sevens, unnamed
d[[3]] <- c(7, 7, 7, 7)
#Add a fourth list, named
d[["student"]] <- c("John", "Peter", "Tome", "Jerry")
```

Add one more list to an exist list (cont'd)

```
d
```

```
## $number
## [1] 1 2 3
##
## $degree
## [1] "master"     "PhD"        "undergrade" "others"     "master"
##
## [[3]]
## [1] 7 7 7 7
##
## $student
## [1] "John"  "Peter" "Tome"  "Jerry"
```

Matrix

A very common mathematical structure that is essential to statistics is a matrix. This is similar to a data.frame in that it is rectangular with rows and columns except that every single element, regardless of column, must be the same type, most commonly all numerics. They also act similarly to vectors with element-by-element addition, multiplication, subtraction, division and equality. The nrow, ncol and dim functions work just like they do for data.frames.

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

Create matrix

Using matrix() function to create matrix

```
#Create 2X3 matrix
A <-  matrix(1:6, nrow=3)
A
```

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

```
nrow(A)
```

```
## [1] 3
```

```
ncol(A)
```

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

Create another matrix

Using matrix() function to create matrix

```
#Create another 2X3 matrix
B <-  matrix(2:7, nrow=3)
B
```

```
##      [,1] [,2]
## [1,]    2    5
## [2,]    3    6
## [3,]    4    7
```

```
nrow(B)
```

```
## [1] 3
```

```
ncol(B)
```

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

Camparison/calculation between two matrix

```
#See if they are equal
A == B
```

```
##       [,1]  [,2]
## [1,] FALSE FALSE
## [2,] FALSE FALSE
## [3,] FALSE FALSE
```

```
#Multiply A by B
A*B
```

```
##      [,1] [,2]
## [1,]    2   20
## [2,]    6   30
## [3,]   12   42
```

Data input to R

As with everything in R, there are numerous ways to get data; the most common is probably reading comma separated values (CSV) files. Of course there are many other options that we will cover as well.

Lander, Jared P.R For Everyone - Advanced Analytics and Graphics. Pearson Education (Us), 2017.

CSV

data source: http://www.jaredlander.com/data/Tomato%20First.csv

```
readurl <- "http://www.jaredlander.com/data/Tomato%20First.csv"
read.csv(readurl)


#For Window users
#be aware of the "/" not "\"
tomato <- read.csv("C:/tomato.csv")
#For Mac user, no need to put disk name
tomato <- read.csv("/Users/tomato.csv")
```

CSV (cont'd)

```
library(tidyverse)
#or
library(readr)

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

Excel data

While Excel may be the world's most popular data analysis tool, it is unfortunately difficult to read Excel data into R. The simplest method would be to use Excel (or another spreadsheet program) to convert the Excel file to a CSV file.

However, if you want to read excel data, you will install some read excel packages, such as readxl.

```
install.packages("readxl")
library(readxl)
Tomato <- read_excel ("Your excel file location")
```

Other statistical tools

Function	Format
read.spss	SPSS
read.dta	Stata
read.ssd	SAS
read.octave	Octave
read.mtp	Minitab
read.systat	Systat