NC STATE UNIVERSITY

Introduction to R for Data Science Part I

Justin Post

Course Schedule

Daily agenda:

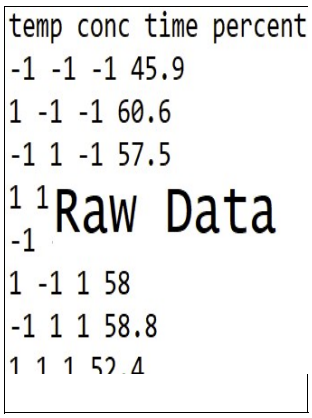
- 9:30-10:40 Session
- 15-minute break
- 10:55-12:05 Session
- 12:05-1:20 Lunch
- 1:20-2:30 Session
- 15-minute break
- 2:45-3:55 Session
- 3:55–4:00 Q&A and Feedback

What is this course about?

Basic use of R for reading, manipulating, and plotting data!

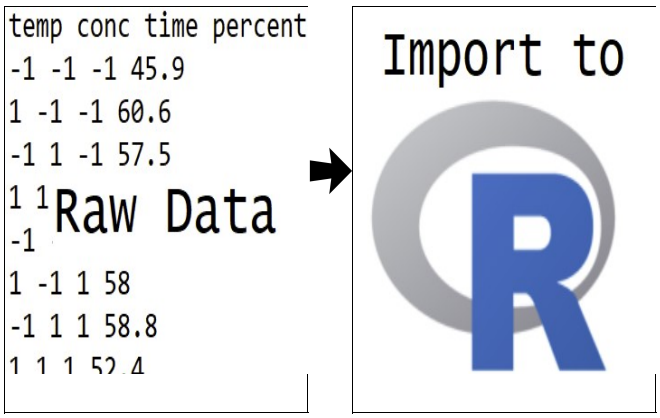
What is this course about?

Basic use of R for reading, manipulating, and plotting data!



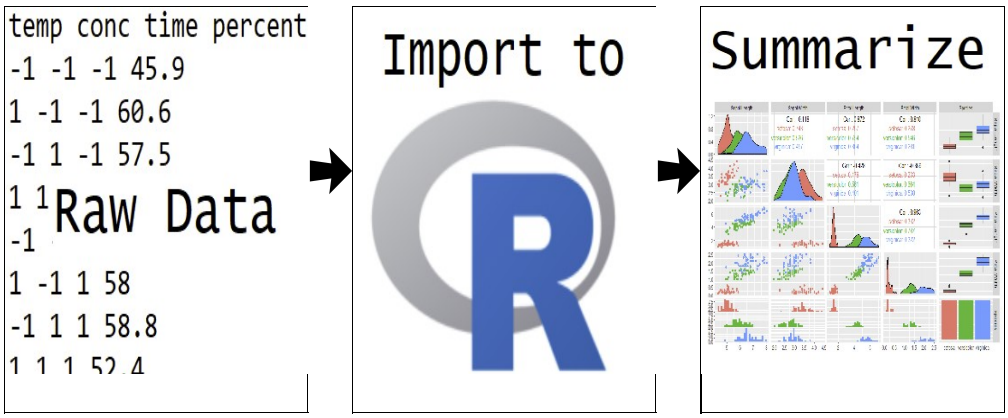
What is this course about?

Basic use of R for reading, manipulating, and plotting data!



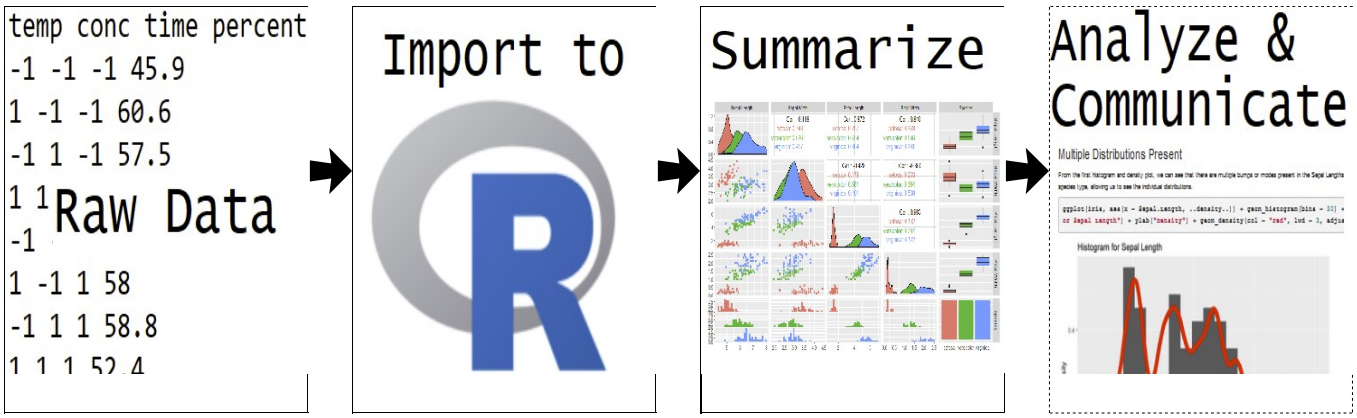
What is this course about?

Basic use of R for reading, manipulating, and plotting data!



What is this course about?

Basic use of R for reading, manipulating, and plotting data!



Where do we start?

Roughly Day 1

- Install R/R studio
- R Studio Interface
- Classes and Objects
- Attributes and Basic Data Object Manipulation
- Reading in Data/Writing Out Data

Roughly Day 2

- Logical Statements and Subsetting/Manipulating Data
- Numerical and Graphical Summaries
- Basic Analyses

Installing R

- Check out the course website
- Info on installing R and R studio available here
- Let's take a few minutes and make sure everyone has these installed and working properly!

If installed, open RStudio, go to 'console' and type

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

Why learn R?

- It's free, open source, available on all major platforms.
- Tons of packages for modeling, visualization, data manipulation, etc.
- Access to the newest methods.
- Great community support (stackoverflow, R-help mailing list, etc.)
- Can *easily* create pdfs, slides, reports, html files, and interactive apps.

Drawbacks of Using R

- Code style differs greatly!
- Confusing! Often many ways to do the same thing
- New code not necessarily verified
- Updates can mess up ‘legacy’ code

RStudio IDE

In RStudio, four main ‘areas’

- Console (& Terminal)
- Scripting and Viewing Window
- Plots/Help (& Files/Packages)
- Environment (& Connections/Git)

Console

- Type code directly into the **console** for evaluation

```
#simple math operations
# <-- is a comment - code not evaluated
3 + 7
```

```
## [1] 10
```

```
10 * exp(3) #exp is exponential function
```

```
## [1] 200.8554
```

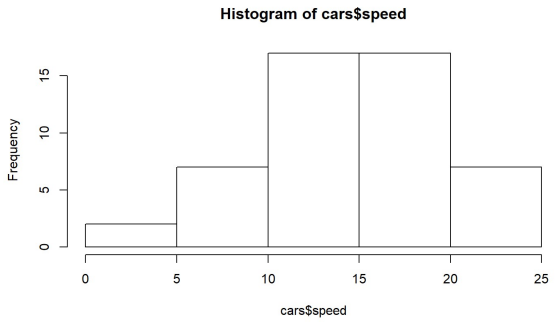
```
log(pi^2) #log is natural log by default
```

```
## [1] 2.28946
```

```
mean(cars$speed)
```

```
## [1] 15.4
```

```
hist(cars$speed)
```



Scripting and Viewing Window

- Usually want to keep code for later use!
- Write code in a ‘script’ and save script (or use markdown - covered later)
- From script can send code to console via:
  - “Run” button (runs current line)
  - CTRL+Enter (PC) or Command+Enter (MAC)
  - Highlight section and do above

Scripting and Viewing Window

- Go to file –> New File –> R Script
- Type `View(cars)` (note capital `V`)
- Type `plot(cars)`
- Submit to console using button or hot key

Plots/Help

- Created plots stored in `Plots` tab
  - Cycle through past plots
  - Easily save
- `Help` tab to learn about R functions
- Type `help(hist)` in the console

Environment

- Store **data/info/function/etc.** in R objects
- Create an R object via <– (recommended) or =

```
#save for later
avg <- (5 + 7 + 6) / 3
#call avg object
avg
```

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

```
#strings (text) can be saved as well
words <- c("Hello there!", "How are you?")
words
```

```
## [1] "Hello there!" "How are you?"
```

Environment

- Look at all current objects with `ls()`

```
ls()
```

```
## [1] "avg"   "words"
```

- `rm()` to remove

```
rm(avg)
ls()
```

```
## [1] "words"
```

Environment

- Built-in objects exist like `letters` and `cars`

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
head(cars, n = 3)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
```

- `data()` shows available built-in datasets

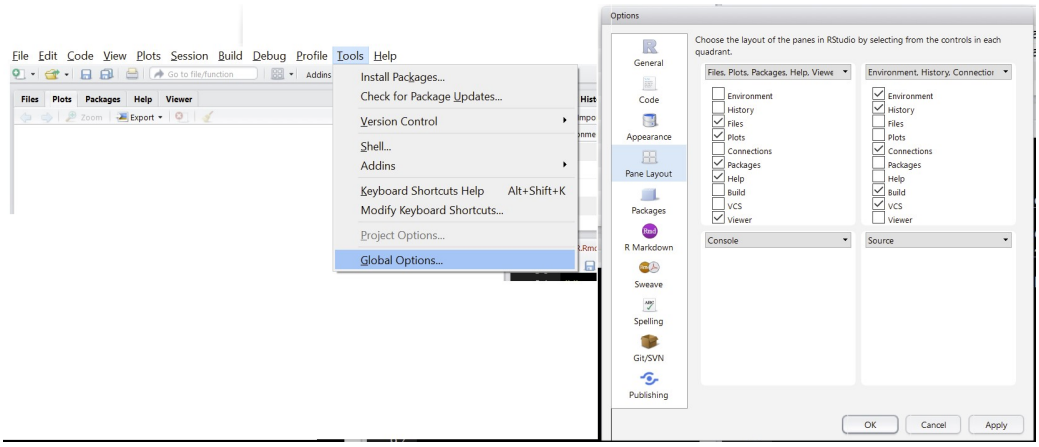
RStudio IDE

Four main ‘areas’

- Console (& Terminal)
- Scripting and Viewing Window
- Plots/Help (& Files/Packages)
- Environment (& Connections/Git)

RStudio IDE

To rearrange panes



RStudio IDE

Other useful global options:

- Appearance
    - font size
    - theme
- Code
    - editing –> soft-wrap
    - display –> show whitespace

R Objects and Classes

- R has strong **O**bject **O**riented **P**rogramming (OOP) tools
- Object: data structure with attributes (class)
- Method: procedures (functions) act on object based on attributes

R Objects and Classes

- R has strong **O**bject **O**riented **P**rogramming (OOP) tools
- Object: data structure with attributes (class)
- Method: procedures (functions) act on object based on attributes
- R functions like `print()` or `plot()` act differently depending on object class

```
class(cars)
```

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

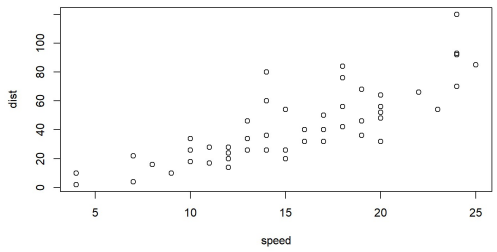
```
class(exp)
```

```
## [1] "function"
```

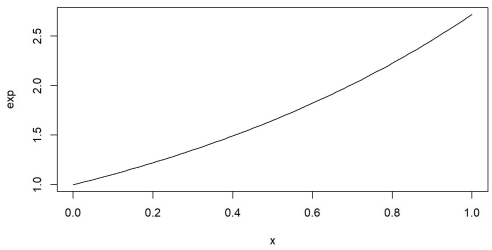
R Objects and Classes

- R has strong **O**bject **O**riented **P**rogramming (OOP) tools
- Object: data structure with attributes (often a ‘class’)
- Method: procedures (often ‘functions’) act on object based on attributes
- R functions like `print()` or `plot()` act differently depending on object class

`plot(cars)`



`plot(exp)`



R Objects and Classes

- Create an R object via <– (recommended) or =
    - allocates memory to object
    - object attributes usually depend on how you created it!

```
vec <- c(1, 4, 10)
class(vec)
```

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

```
fit <- lm(dist ~ speed, data = cars)
class(fit)
```

```
## [1] "lm"
```

Investigating Objects

Many functions to help understand an R Object

- `class()`
- describes the `class` attribute of an R object

```
class(cars)
```

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

Investigating Objects

Many functions to help understand an R Object

- `typeof()`
- determines the (R internal) type or storage mode of any object

```
typeof(cars)
```

```
## [1] "list"
```

Investigating Objects

Many functions to help understand an R Object

- `str()`
- compactly displays the internal structure of an R object

```
str(cars)
```

```
## 'data.frame':    50 obs. of  2 variables:
##  $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
##  $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

Recap & What's next?!

Create an R Object with <-

Many functions to help understand an R Object

- `class()`
- `typeof()`
- `str()`

Recap & What's next?!

Create an R Object with `<-`

Many functions to help understand an R Object

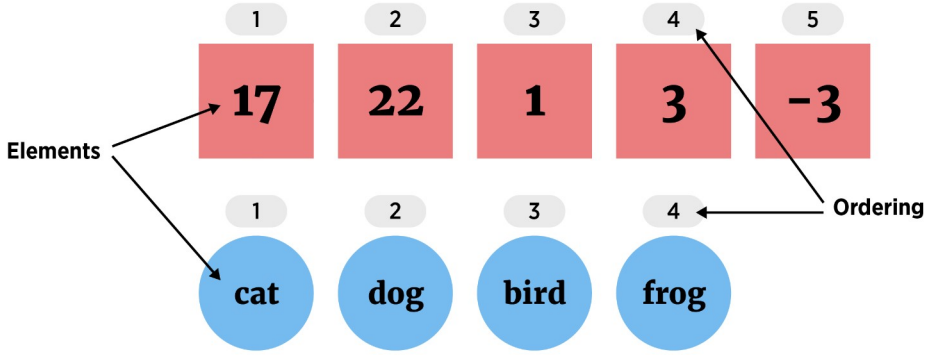
- `class()`
- `typeof()`
- `str()`

Common data structures

1. Atomic Vector (1d)
2. Matrix (2d)
3. Array (nd) (not covered)
4. Data Frame (2d)
5. List (1d)

Vector

1. Atomic Vector (1D group of elements with an ordering)



- Elements must be same ‘type’
    - numeric (integer or double), character, or logical

Vector

1. Atomic Vector (1D group of elements with an ordering)

- Create with `c()` function ('combine')

```
#vectors (1 dimensional) objects
x <- c(17, 22, 1, 3, -3)
y <- c("cat", "dog", "bird", "frog")
x
```

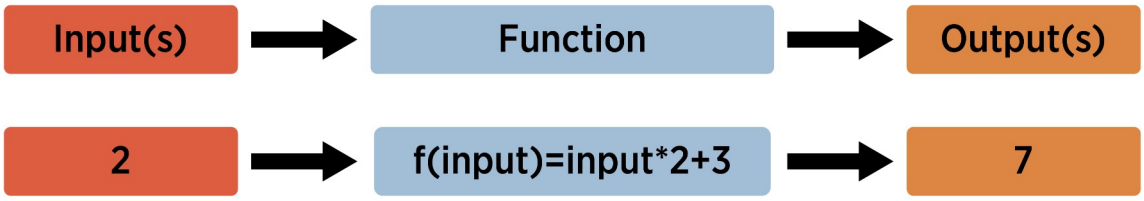
```
## [1] 17 22  1  3 -3
```

```
y
```

```
## [1] "cat"  "dog"  "bird" "frog"
```

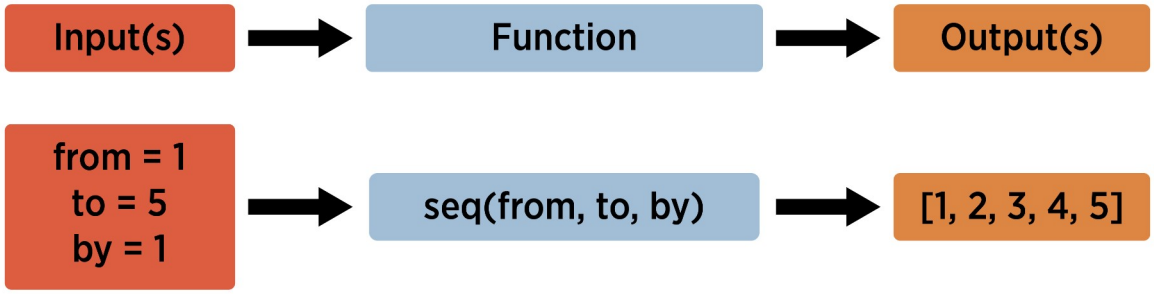
Vector

- Many ‘functions’ output a numeric vector
- Function concept:



Vector

- Many 'functions' output a numeric vector
- Ex: `seq()`
  - Inputs = from, to, by (among others)
  - Output = a sequence of numbers



Vector

```
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
length.out = NULL, along.with = NULL, ...)
```

```
v <- seq(from = 1, to = 5, by = 1)
v
```

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

Vector

```
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
length.out = NULL, along.with = NULL, ...)
```

```
v <- seq(from = 1, to = 5, by = 1)
v
```

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

```
str(v)
```

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

- `num` says it is numeric
- `[1:5]` implies one dimensional with elements 1, 2, 3, 4, 5

Vector

Shorthand `seq()` with `:`

```
1:20
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Vector

Shorthand `seq()` with `:`

- R generally does elementwise math

```
1:20/20
```

```
##  [1] 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75
## [16] 0.80 0.85 0.90 0.95 1.00
```

```
1:20 + 1
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
```

Help Files

- Functions are ubiquitous in R!
- To find out about a function's arguments use `help()`
- Understanding the syntax in the help files is key!
- Ex: Can create randomly generated values in any interval:
  - `help(runif)`

Vector

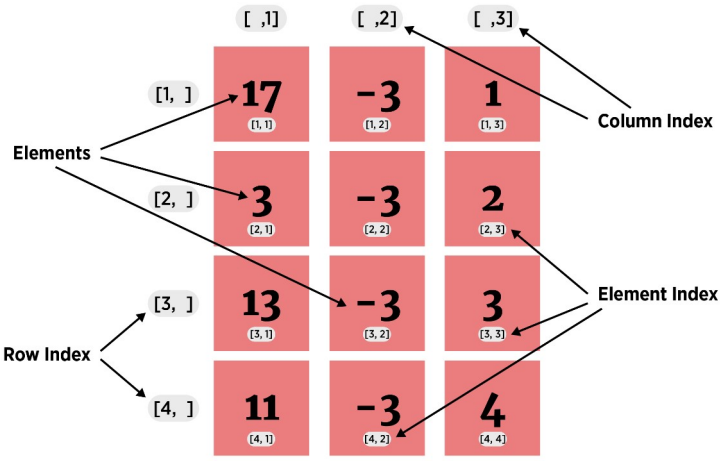
1. Atomic Vector (1D group of elements with an ordering)

- Vectors useful to know about
- Not usually useful for a dataset
- Often consider as ‘building blocks’ for other data types

Matrix

2. Matrix (2D data structure)

- (think) columns are vectors of the same **type and length**



Matrix

2. Matrix (2D data structure)

- (think) columns are vectors of the same **type and length**

```
#populate vectors
x <- c(17, 3, 13, 11)
y <- rep(-3, times = 4)
z <- 1:4
```

Matrix

2. Matrix (2D data structure)

- (think) columns are vectors of the same **type and length**

```
#populate vectors
x <- c(17, 3, 13, 11)
y <- rep(-3, times = 4)
z <- 1:4
```

```
#check 'type'
is.numeric(x)
```

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

```
is.numeric(y)
```

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

```
is.numeric(z)
```

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

Matrix

2. Matrix (2D data structure)

- (think) columns are vectors of the same **type and length**

```
#populate vectors
x <- c(17, 3, 13, 11)
y <- rep(-3, times = 4)
z <- 1:4
```

```
#check 'type'
is.numeric(x)
```

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

```
is.numeric(y)
```

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

```
is.numeric(z)
```

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

```
#check 'length'
length(x)
```

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

```
length(y)
```

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

```
length(z)
```

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

Matrix

2. Matrix (2D data structure)

- (think) columns are vectors of the same **type and length**
- Create with `matrix()` function (see help)

Matrix

2. Matrix (2D data structure)

- (think) columns are vectors of the same **type and length**
- Create with `matrix()` function (see help)

```
#populate vectors
x <- c(17, 3, 13, 11)
y <- rep(-3, times = 4)
z <- 1:4
#combine in a matrix
matrix(c(x, y, z), ncol = 3)
```

```
##      [,1] [,2] [,3]
## [1,]   17   -3    1
## [2,]    3   -3    2
## [3,]   13   -3    3
## [4,]   11   -3    4
```

Matrix

2. Matrix (2D data structure)

- (think) columns are vectors of the same **type and length**
- Create with `matrix()` function

```
x <- c("Hi", "There", "Friend", "!")
y <- c("a", "b", "c", "d")
z <- c("One", "Two", "Three", "Four")
is.character(x)
```

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

```
matrix(c(x, y, z), nrow = 6)
```

```
##      [,1]     [,2]
## [1,] "Hi"     "c"
## [2,] "There"  "d"
## [3,] "Friend" "One"
## [4,] "!"      "Two"
## [5,] "a"      "Three"
## [6,] "b"      "Four"
```

Matrix

2. Matrix (2D data structure)

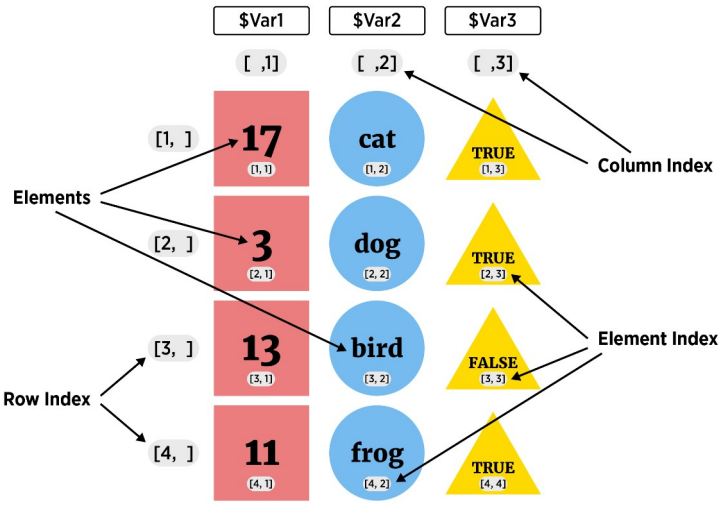
- (think) columns are vectors of the same **type and length**
- Useful for some data but often some numeric and some character variables:

brand	tar	nicotine	weight	co
Alpine	14.1	0.86	0.9853	13.6
Benson	16.0	1.06	1.0938	16.6
CamelLights	8.0	0.67	0.9280	10.2
Carlton	4.1	0.40	0.9462	5.4
Chesterfield	15.0	1.04	0.8885	15.0
GoldenLights	8.8	0.76	1.0267	9.0
Kent	12.4	0.95	0.9225	12.3
Kool	16.6	1.12	0.9372	16.3
L&M	14.9	1.02	0.8858	15.4
LarkLights	13.7	1.01	0.9643	13.0

Data Frame

4. Data Frame (2D data structure)

- collection (list) of *vectors* of the same **length**



Data Frame

4. Data Frame (2D data structure)

- collection (list) of *vectors* of the same **length**
- Create with `data.frame()` function

```
x <- c("a", "b", "c", "d", "e", "f")
y <- c(1, 3, 4, -1, 5, 6)
z <- 10:15
data.frame(x, y, z)
```

```
##   x  y  z
## 1 a  1 10
## 2 b  3 11
## 3 c  4 12
## 4 d -1 13
## 5 e  5 14
## 6 f  6 15
```

Data Frame

4. Data Frame (2D data structure)

- collection (list) of *vectors* of the same **length**
- Create with `data.frame()` function

```
data.frame(char = x, data1 = y, data2 = z)
```

```
##   char data1 data2
## 1    a     1    10
## 2    b     3    11
## 3    c     4    12
## 4    d    -1    13
## 5    e     5    14
## 6    f     6    15
```

- char, data1, and data2 become the variable names for the data frame

Data Frame

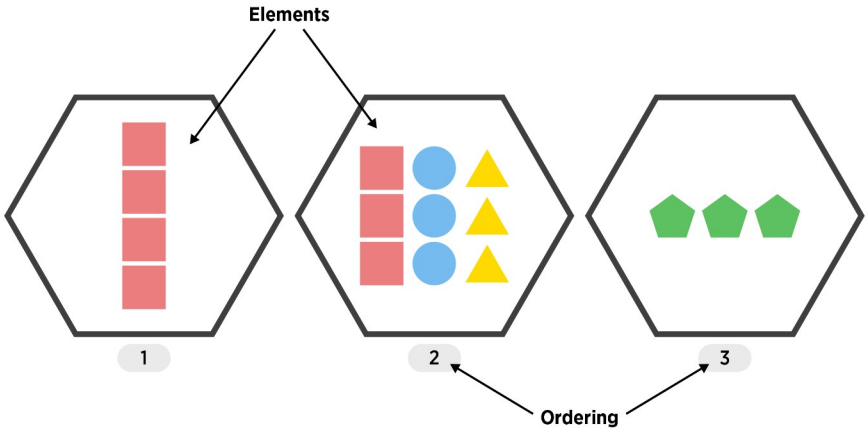
4. Data Frame (2D data structure)

- collection (list) of *vectors* of the same **length**
- Create with `data.frame()` function
- Perfect for most data sets!
- Most functions that read 2D data store it as a data frame

List

5. List (1D group of objects with ordering)

- a vector that can have differing elements



List

5. List (1D group of objects with ordering)

- a vector that can have differing elements
- Create with `list()`

```
list(1:3, rnorm(2), c("!", "?"))
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] 0.1832123 0.2735573
##
## [[3]]
## [1] "!" "?"
```

List

5. List (1D group of objects with ordering)

- Add names to the list elements

```
list(seq = 1:3, normVals = rnorm(2), punctuation = c("!", "?"))
```

```
## $seq
## [1] 1 2 3
##
## $normVals
## [1] -0.8738642  1.4966295
##
## $punctuation
## [1] "!" "?"
```

List

5. List (1D group of objects with ordering)

- a vector that can have differing elements
- Create with `list()`
- More flexible than a Data Frame!
- Useful for more complex types of data

Recap!

Dimension	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data Frame

- For most data analysis you'll use data frames!
- Next up: How do we access/change parts of our objects?

Activity

- **Objects and Common Classes Activity** instructions available on web
- We'll send you to breakout rooms
- One TA or instructor in each room to help out
- Feel free to ask questions about anything you didn't understand as well!

Basic Data Manipulation

- How do we access different parts of our object?

Basic Data Manipulation

- **How do we access different parts of our object?**
- For data may want
    - One element
    - Certain columns
    - Certain rows

Basic Data Manipulation

Atomic Vectors (1D)

- Return elements using square brackets `[]`

```
letters #built-in vector
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
letters[1] #R starts counting at 1!
```

```
## [1] "a"
```

```
letters[26]
```

```
## [1] "z"
```

Basic Data Manipulation

Atomic Vectors (1D)

- Can 'feed' in a vector of indices to return

```
letters[1:4]
```

```
## [1] "a" "b" "c" "d"
```

```
letters[c(5, 10, 15, 20, 25)]
```

```
## [1] "e" "j" "o" "t" "y"
```

```
x <- c(1, 2, 5); letters[x]
```

```
## [1] "a" "b" "e"
```

Basic Data Manipulation

Atomic Vectors (1D)

- Return elements using square brackets `[]`
- Can 'feed' in a vector of indices to return
- Use negative indices to return without

```
letters[-(1:4)]
```

```
##  [1] "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w"
## [20] "x" "y" "z"
```

```
x <- c(1, 2, 5); letters[-x]
```

```
##  [1] "c" "d" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v"
## [20] "w" "x" "y" "z"
```

Basic Data Manipulation

Matrices (2D)

- Use square brackets with a comma `[ , ]`
- Notice default row and column names!

```
mat <- matrix(c(1:4, 20:17), ncol = 2)
mat
```

```
##      [,1] [,2]
## [1,]    1   20
## [2,]    2   19
## [3,]    3   18
## [4,]    4   17
```

Basic Data Manipulation

Matrices (2D)

- Use square brackets with a comma `[ , ]`

```
mat
```

```
##      [,1] [,2]
## [1,]    1   20
## [2,]    2   19
## [3,]    3   18
## [4,]    4   17
```

```
mat[c(2, 4), ]
```

```
##      [,1] [,2]
## [1,]    2   19
## [2,]    4   17
```

```
mat[, 1]
```

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

```
mat[2, ]
```

```
## [1]  2 19
```

```
mat[2, 1]
```

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

Basic Data Manipulation

Matrices (2D)

- Can give columns names
- `help(matrix)` can show us how!

Basic Data Manipulation

Matrices (2D)

- Can use columns names to subset

```
mat <- matrix(c(1:4, 20:17), ncol = 2,
          dimnames = list(NULL,
                c("First", "Second"))
        )
mat
```

```
##      First Second
## [1,]     1     20
## [2,]     2     19
## [3,]     3     18
## [4,]     4     17
```

```
mat[, "First"]
```

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

Basic Data Manipulation

Matrices (2D)

- Use square brackets with a comma `[ , ]`
- Can use columns names to subset
- Negative still removes but won't work with column name

```
mat[-c(1,3), -"First"]
```

```
## Error in -"First": invalid argument to unary operator
```

```
mat[-c(1,3), "First"]
```

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

Basic Data Manipulation

Data Frames (2D)

- Consider 'built-in' `iris` data frame

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Basic Data Manipulation

Data Frames (2D)

- Data Frame is 2D similar to a matrix - access similarly!
- Use square brackets with a comma `[ , ]`

```
iris[1:4, 2:4]
```

```
##   Sepal.Width Petal.Length Petal.Width
## 1         3.5          1.4         0.2
## 2         3.0          1.4         0.2
## 3         3.2          1.3         0.2
## 4         3.1          1.5         0.2
```

Basic Data Manipulation

Data Frames (2D)

- Data Frame is 2D similar to a matrix - access similarly!
- Use square brackets with a comma `[ , ]`

```
iris[1, ]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
```

Basic Data Manipulation

Data Frames (2D)

- Can use columns names to subset

```
iris[ , c("Sepal.Length", "Species")]
```

```
##     Sepal.Length    Species
## 1            5.1     setosa
## 2            4.9     setosa
## 3            4.7     setosa
## 4            4.6     setosa
## 5            5.0     setosa
## 6            5.4     setosa
## 7            4.6     setosa
## 8            5.0     setosa
## 9            4.4     setosa
## 10           4.9     setosa
## 11           5.4     setosa
## 12           4.8     setosa
## 13           4.8     setosa
## 14           4.3     setosa
## 15           5.8     setosa
## 16           5.7     setosa
## 17           5.4     setosa
## 18           5.1     setosa
## 19           5.7     setosa
## 20           5.1     setosa
## 21           5.4     setosa
## 22           5.1     setosa
## 23           4.6     setosa
## 24           5.1     setosa
## 25           4.8     setosa
## 26           5.0     setosa
## 27           5.0     setosa
## 28           5.2     setosa
## 29           5.2     setosa
## 30           4.7     setosa
## 31           4.8     setosa
## 32           5.4     setosa
## 33           5.2     setosa
## 34           5.5     setosa
## 35           4.9     setosa
## 36           5.0     setosa
## 37           5.5     setosa
## 38           4.9     setosa
```

Basic Data Manipulation

Data Frames (2D)

- Dollar sign allows easy access to a single column!

```
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

Basic Data Manipulation

Data Frames (2D)

- Dollar sign allows easy access to a single column!
- Most used method for accessing a single variable
- RStudio fills in options.
    - Type `iris$`
    - If no choices - hit tab
    - Hit tab again to choose

Basic Data Manipulation

Data Frames (2D)

- Data Frame is 2D similar to a matrix - access similarly!
- Use square brackets with a comma `[ , ]`
- Can use columns names to subset
- Dollar sign allows easy access to a single column!

Basic Data Manipulation

Lists (1D)

- Use single square brackets `[ ]` for multiple list elements

```
x <- list("HI", c(10:20), 1)
x
```

```
## [[1]]
## [1] "HI"
##
## [[2]]
##  [1] 10 11 12 13 14 15 16 17 18 19 20
##
## [[3]]
## [1] 1
```

Basic Data Manipulation

Lists (1D)

- Use single square brackets `[ ]` for multiple list elements

```
x <- list("HI", c(10:20), 1)
x[2:3]
```

```
## [[1]]
##  [1] 10 11 12 13 14 15 16 17 18 19 20
##
## [[2]]
## [1] 1
```

Basic Data Manipulation

Lists (1D)

- Use double square brackets `[[ ]]` (or `[ ]`) for single list element

```
x <- list("HI", c(10:20), 1)
x[1]
```

```
## [[1]]
## [1] "HI"
```

```
x[[1]]
```

```
## [1] "HI"
```

```
x[[2]]
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

```
x[[2]][4:5]
```

```
## [1] 13 14
```

Basic Data Manipulation

Lists (1D)

- If named list elements, can use $

```
x <- list("HI", c(10:20), 1)
str(x)
```

```
## List of 3
##  $ : chr "HI"
##  $ : int [1:11] 10 11 12 13 14 15 16 17 18 19 ...
##  $ : num 1
```

```
x <- list(First = "Hi", Second = c(10:20), Third = 1)
x$Second
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

Basic Data Manipulation

Lists & Data Frames

- Connection: Data Frame = *List* of equal length vectors

```
str(x)
```

```
## List of 3
##  $ First : chr "Hi"
##  $ Second: int [1:11] 10 11 12 13 14 15 16 17 18 19 ...
##  $ Third : num 1
```

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Basic Data Manipulation

Lists & Data Frames

- Connection: Data Frame = *List* of equal length vectors

```
typeof(x)
```

```
## [1] "list"
```

```
typeof(iris)
```

```
## [1] "list"
```

Basic Data Manipulation

Lists & Data Frames

- Connection: Data Frame = *List* of equal length vectors

```
iris[[2]]
```

```
##   [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9 3.5
##  [19] 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2 3.1 3.2
##  [37] 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2 3.2 3.1 2.3
##  [55] 2.8 2.8 3.3 2.4 2.9 2.7 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7 2.2 2.5 3.2 2.8
##  [73] 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0 3.4 3.1 2.3 3.0 2.5
##  [91] 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7 3.0 2.9 3.0 3.0 2.5 2.9
## [109] 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6 2.2 3.2 2.8 2.8 2.7 3.3 3.2
## [127] 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2
## [145] 3.3 3.0 2.5 3.0 3.4 3.0
```

Partial Matching

Lists & Data Frames

With `[[` or `$` partial matching can be used

```
iris$Sp
```

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     setosa
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     versicolor versicolor versicolor versicolor
##  [55] versicolor versicolor versicolor versicolor versicolor versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor
##  [67] versicolor versicolor versicolor versicolor versicolor versicolor
##  [73] versicolor versicolor versicolor versicolor versicolor versicolor
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor versicolor versicolor versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor virginica  virginica
## [103] virginica  virginica  virginica  virginica  virginica  virginica
## [109] virginica  virginica  virginica  virginica  virginica  virginica
## [115] virginica  virginica  virginica  virginica  virginica  virginica
## [121] virginica  virginica  virginica  virginica  virginica  virginica
## [127] virginica  virginica  virginica  virginica  virginica  virginica
## [133] virginica  virginica  virginica  virginica  virginica  virginica
## [139] virginica  virginica  virginica  virginica  virginica  virginica
## [145] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

Partial Matching

Lists & Data Frames

With `[[` or `$` partial matching can be used

```
iris[["Petal.Len", exact = FALSE]]
```

```
##   [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2 1.5 1.3 1.4
##  [19] 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4 1.5 1.2
##  [37] 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4 4.7 4.5 4.9 4.0
##  [55] 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.0 4.7 3.6 4.4 4.5 4.1 4.5 3.9 4.8 4.0
##  [73] 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.1 4.5 4.5 4.7 4.4 4.1 4.0
##  [91] 4.4 4.6 4.0 3.3 4.2 4.2 4.2 4.3 3.0 4.1 6.0 5.1 5.9 5.6 5.8 6.6 4.5 6.3
## [109] 5.8 6.1 5.1 5.3 5.5 5.0 5.1 5.3 5.5 6.7 6.9 5.0 5.7 4.9 6.7 4.9 5.7 6.0
## [127] 4.8 4.9 5.6 5.8 6.1 6.4 5.6 5.1 5.6 6.1 5.6 5.5 4.8 5.4 5.6 5.1 5.1 5.9
## [145] 5.7 5.2 5.0 5.2 5.4 5.1
```

Basic Data Manipulation

Lists (1D)

- Use single square brackets `[ ]` for multple list elements
- Use double square brackets `[[ ]]` (or `[ ]`) for single list element
- If named list elements, can use $

Recap!

Dimension	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data Frame

Basic access via

- Atomic vectors - `x[ ]`
- Matrices - `x[ , ]`
- Data Frames - `x[ , ]` or `x$name`
- Lists - `x[ ]`, `x[[ ]]`, or `x$name`

Activity

- **Attributes and Basic Data Manipulation Activity** instructions available on web
- We'll send you to breakout rooms
- One TA or instructor in each room to help out
- Feel free to ask questions about anything you didn't understand as well!